Lecture Notes in Computer Science 1862

Peter G. Clote
Helmut Schwichtenberg (Eds.)

Computer Science Logic

14th International Workshop, CSL 2000
Annual Conference of the EACSL
Fischbachau, Germany, August 2000
Proceedings

Springer

Lecture Notes in Computer Science 1862
Edited by G. Goos, J. Hartmanis and J. van Leeuwen

Springer
Berlin
Heidelberg
New York
Barcelona
Hong Kong
London
Milan
Paris
Singapore
Tokyo

Peter G. Clote Helmut Schwichtenberg (Eds.)

Computer Science Logic

14th International Workshop, CSL 2000
Annual Conference of the EACSL
Fischbachau, Germany, August 21 - 26, 2000
Proceedings

Springer

Volume Editors

Peter G. Clote
Boston College
Department of Computer Science and Department of Biology
Fulton Hall 410B, Chestnut Hill, MA 02467, USA
E-mail: clote@bc.edu

Helmut Schwichtenberg
Ludwig-Maximilian-Universität München
Mathematisches Institut
Theresienstr. 39, 80333 München, Germany
E-mail: schwicht@rz.mathematik.uni-muenchen.de

Cataloging-in-Publication Data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Computer science logic : 14th international workshop ; proceedings /
CSL 2000, Fischbachau, Germany, August 21 - 26, 2000. Peter Clote ;
Helmut Schwichtenberg (ed.). - Berlin ; Heidelberg ; New York ;
Barcelona ; Hong Kong ; London ; Milan ; Paris ; Singapore ; Tokyo :
Springer, 2000
(Annual Conference of the EACSL . . . ; 9)
(Lecture notes in computer science ; Vol. 1862)
ISBN 3-540-67895-6

CR Subject Classification (1998): F.4, I.2.3-4, F.3

ISSN 0302-9743
ISBN 3-540-67895-6 Springer-Verlag Berlin Heidelberg New York

Springer-Verlag Berlin Heidelberg New York
a member of BertelsmannSpringer Science+Business Media GmbH

Printed in Germany

Typesetting: Camera-ready by author, data conversion by Steingräber Satztechnik GmbH, Heidelberg
Printed on acid-free paper SPIN 10722230 06/3142 5 4 3 2 1 0

Preface

CSL is the annual conference of the European Association for Computer Science Logic (EACSL). CSL 2000 is the 14th such annual conference, thus witnessing the importance and sustained international interest in the application of methods from mathematical logic to computer science. The current conference was organized by the Mathematics Institute and the Computer Science Institute of the Ludwig-Maximilians-Universität München (LMU), with generous financial support from the Deutsche Forschungsgemeinschaft, Forschungsinstitut für angewandte Softwaretechnologie (FAST e.V.), Münchener Universitätsgesellschaft e.V., and Siemens AG. Our sponsors' generosity enabled, among other things, stipends for the financial support of students as well as of researchers from Eastern Europe.

Topics in the call for papers for CSL 2000 included: automated deduction and interactive theorem proving, categorical logic and topological semantics, constructive mathematics and type theory, domain theory, equational logic and term rewriting, finite model theory, database theory, higher order logic, lambda and combinatory calculi, logical aspects of computational complexity, logical foundations of programming paradigms, logic programming and constraints, linear logic, modal and temporal logics, model checking, program extraction, program logics and semantics, program specification, transformation and verification. The invited speakers were: Moshe Vardi (Houston), Paul Beame (Washington), Andreas Blass (Ann Arbor), Egon Börger (Pisa), Yuri Gurevich (Redmond), Bruno Poizat (Lyons), Wolfram Schulte (Redmond), Saharon Shelah (Jerusalem), and Colin Sterling (Edinburgh). Special thanks to Moshe Vardi for being willing to speak in the place of Miklós Ajtai (Almaden), who could not attend the meeting.

The day of 24 August 2000, during the week-long CSL 2000 meeting, was reserved for the *Gurevich Symposium*, a special, one-day tribute to the scientific contributions of Professor Yuri Gurevich, at the occasion of his 60th birthday. Many of the previously listed invited speakers delivered a talk at the Gurevich Symposium. As editors of the proceedings, we would like to dedicate this volume of the Springer Lecture Notes in Computer Science to Professor Gurevich.

We would like to thank the Program Committee, the referees, and our sponsors for making this conference possible. A special thanks goes to the Organizing Committee for its professional work ranging from practical to editorial matters. Finally, thanks to the assistance of the authors, who formatted their articles using LaTeXwith Springer macros. This allowed rapid production of the proceedings first distributed at the conference.

May 2000 Peter Clote and Helmut Schwichtenberg

Sponsors

Deutsche Forschungsgemeinschaft
Forschungsinstitut für angewandte Softwaretechnologie (FAST e.V.)
Münchener Universitätsgesellschaft e.V.
Siemens AG.

Organizing Committee

Thorsten Altenkirch
Rolf Backofen
Peter Clote
Hans Leiß
Ralph Matthes
Martin Ruckert
Helmut Schwichtenberg.

Program Committee

Peter Clote (Boston, co-chair)
Kevin Compton (Ann Arbor)
Stephen Cook (Toronto)
Laurent Fribourg (Cachan)
Erich Grädel (Aachen)
Gerhard Jäger (Bern)
Klaus Keimel (Darmstadt)
Jan Willem Klop (Nijmegen)
Jan Krajíček (Praha)
Daniel Leivant (Bloomington)
Tobias Nipkow (Munich)
Helmut Schwichtenberg (Munich, co-chair)
Moshe Vardi (Houston).

Referees

S. Abramsky	Th. Altenkirch	R. Amadio	A. Asperti
S. Awodey	A. Beckmann	S. Bellantoni	S. Berardi
U. Berger	G. Bezhanishvili	M. Bezem	M. Bidoit
N. Bjørner	H. Blair	F. de Boer	T. Borghuis
K. Bruce	A. Bucciarelli	W. Buchholz	P. Bürgisser
A. Cichon	P. Clote	H. Comon	K. Compton
T. Coquand	S. Cosmadakisos	K. Crary	G. Curi
D. van Dalen	V. Danos	P. Darondeau	A. Dawar
O. Deiser	S. Demri	P. Dybjer	R. Dyckhoff
L. Errington	M. Escardo	S. Etalle	A. Felty
Ch. Fermüller	M. Fiore	W.-J. Fokkink	I. Gnaedig
R. Goldblatt	Y. Gordon	R. Gore	G. Governatori
E. Grädel	M. Große-Rhode	S. Guerrini	P. Gumm
P. Hájek	S. Heilmann	H. Herbelin	Ch. Herrmann
R. Hinze	M. Hofmann	F. Honsell	N. Immerman
F. Jacquemard	R. Jagadeesan	F. Joachimski	A. Jung
B. Kapron	M. Kegelmann	J.W. Klop	J. Krajíček
U. Kohlenbach	Ch. Kreitz	S. Kreutzer	O. Kullmann
G. Kuper	H. Lauri	C. Lautemann	D. Leivant
S. Lindell	M. Loebl	B. Long	J. Makowsky
L. Maksimova	P. Malacaria	C. Marché	R. Matthes
J. Meseguer	W. Meyer-Viol	D. Miller	A. Momigliano
G. Nadathur	T. Nipkow	H. de Nivelle	D. Norman
J. Nurmonen	D. von Oheimb	P. Ölveczky	L. Ong
V. van Oostrom	M. Otto	C. Palamidessi	E. Palmgren
Ch. Paulin	A. Poetzsch-Heffter	E. Poll	P. Pudlák
F. van Raamsdonk	L. Regnier	D. Remy	L. Roversi
M. Ruckert	G. Salzer	V. Sazonov	Ch. Schallhart
K. Schlechta	M. Schmidt-Schauß	P. Schnoebelen	P. Schroeder-Heister
H. Schwichtenberg	L. Segoufin	P. Selinger	A. Setzer
S. Soloviev	R. Stärk	C. Stone	T. Strahm
T. Streicher	F. Tang	A. Tarlecki	S. Tobies
J. Tucker	S. Tupailo	L. Vandeurzen	M. Vardi
H. Veith	A. Visser	A. Voronkov	G. Voutsadakis
I. Walukiewicz	C. Weidenbach	T. Wilke	F. Wolter

Yuri Gurevich: The Evolution of a Research Life from Algebra through Logic to Computer Science
by Egon Börger (University of Pisa)

Yuri Gurevich is the man whose life embraces three worlds—Russia, Israel, and United States—and whose research spans three disciplines—algebra, logic, and computer science—all of which shaped the 20th century. In each of these research areas Gurevich set milestones and became a leading figure. Indeed the outstanding constant in his life is his courage and strength to think about the *fundamentals* which underly problems of *impact* in the field.

He was born on May 7, 1940, in Nikolayev (Ukraine) and was moved by life through Stalingrad (41-42), Chimkent (Uzbekistan, 42-44), Cheliabinsk (44-59, school and polytechnic) to Sverdlovsk where he studied, graduated, and taught at the Ural University (59-64, 65-71) and where he married Zoe, a student of mathematics and later system programmer who has given him two daughters, Hava and Naomi, and accompanies his life.

Gurevich started his research in algebra where he became famous through his work on ordered abelian groups. For his diploma in 1962 he solved [1][1] one of the problems which was listed as open in Petr Kontorovich's algebra seminar. Gurevich learned logic from Kleene's *Introduction to Metamathematics* and in 1962 heard about Tarski's program of classifying elementary theories into decidable and undecidable. A year after Tarski and Smielev announced the decidability of the elementary theory of ordered abelian groups but then found an error in their proof, Gurevich proved the decidability of the first-order theory of ordered abelian groups [3], which became his PhD thesis (1964) and made him an assistant professor at the University of Krasnoyarsk (64-65).

Since the theorems in the then known algebra of ordered abelian groups were not first-order, and the standard extensions of classical first-order logic (like monadic second-order logic) give rise to undecidable theories, Gurevich wondered whether there is a logic that fits ordered abelian groups, so that the corresponding theory expresses most of the relevant algebra but yet is manageable and hopefully decidable. He proved that the extension of the elementary theory of ordered abelian groups with quantification over so-called convex subgroups, even though it is much richer than the elementary theory and expresses virtually all known algebra of ordered abelian groups, is not only decidable [25], but allows the elimination of the elementary quantifiers. When in 1973, via Krasnodar (71-72) and Tbilisi (Georgia, 72-73), Gurevich emigrated to Israel (Beer Sheva, 74-82), he met Saharon Shelah, studied Shelah's seminal paper on the theory of order (Annals of Mathematics, 1975), and solved in [26, 27] most of its numerous conjectures, which led to a still ongoing fruitful collaboration between the two logicians (see the survey [64]).

For Hilbert's *Entscheidungsproblem*, one of the major themes of mathematical logic in the twentieth century, Gurevich [6] resolved the most difficult of the

[1] The numbers in brackets refer to the Annotated Publication List at http://research.microsoft.com/~gurevich/, except where marked by *reference below*.

prefix-predicate classes and thereby completed the prefix-predicate classification of the fragments of the restricted predicate calculus as decidable or undecidable. He found a general explanation of the classifiability phenomenon [13], confirmed it for the classification of fragments with function symbols [18], and conjectured the classification of fragments with equality and at least one function symbol, one of the most difficult problems in the area which was later proved by Saharon Shelah. Details can be found in [2] (reference below).

The year 1982, when the University of Michigan appointed Gurevich, marks the beginning of his commitment to computer science and of his close collaboration with Andreas Blass from the mathematics department there. Gurevich shaped the two emerging fields of finite model theory and of average case complexity. It started with his first talk to a computer science conference [41], where Gurevich saw Moshe Vardi applying the definability theorem of first-order logic to databases, which were assumed to be possibly infinite - and immediately worried whether such classical theorems would remain valid if only finite databases were allowed. The answer turned out to be negative [60], the counter-example to Lyndon's interpolation theorem [72] gave a uniform sequence of constant-depth polynomial-size (functionally) monotone boolean circuits not equivalent to any (however nonuniform) sequence of constant-depth polynomial-size positive boolean circuits. Other landmark contributions to finite model theory are the characterization of primitive recursive (resp. recursive) functions over finite structures as log-space (resp. polynomial time) computable [51], the characterization of the inflationary fixed-point extension of first-order logic as equi-expressive with its least fixed-point extension on finite structures [70], the boundedness of every first-order expressible datalog query [83], etc.

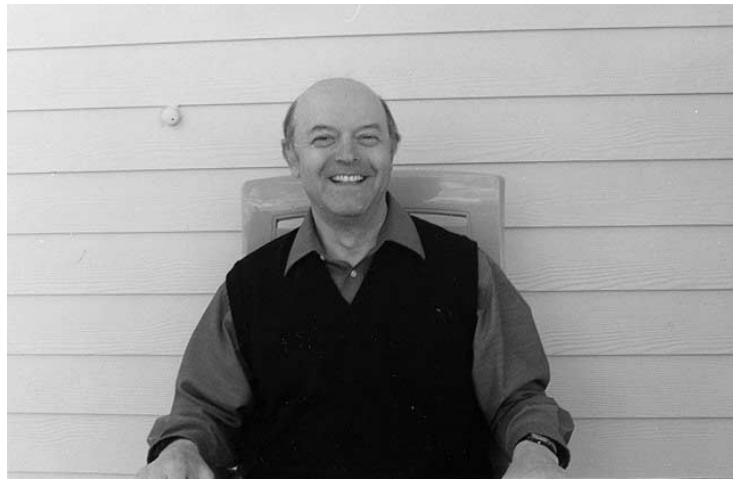
Gurevich's contributions to complexity theory are no less important. He solved special but important cases of NP-hard problems [54], on time-space trade-offs [87], on linear time [82], and critically analyzed some non-traditional approaches [81,80], but above all we see Gurevich work here on average case complexity, side by side with Leonid Levin. An NP-hard problem, when equipped with a probability distribution, may become easy. For example, for random graphs with n vertices and a fixed edge probability, the algorithm of [71] solves the Hamiltonian Circuit Problem in average time $O(n)$. In 1984, Leonid Levin generalized the NP-completeness theory to such distributional (DNP) problems and constructed one DNP problem that was hard in the average case. [76] provides new hard cases and shows that Levin's original deterministic reductions are inadequate, which led Levin and Venkatesan to define more powerful randomizing reductions. [88,93,94,96,97] contains pioneering work towards establishing the average-case intractability of important problems.

Reconsidering Turing's thesis and the fundamental problem of semantics of programming languages led Gurevich to his epochal concept of Abstract State Machines [74,92,103,141]. It has already triggered hundreds of publications, in finite model theory [109,120,135], in complexity theory [118,121], and in numerous areas of applied computer science, e.g. programming languages, protocols, architectures, and embedded control software (see the survey in [1] (reference

below) and [77,89,98,106,107,111,116,117,119,121,122,137-140]); even more importantly it is changing the way we think about high-level software design and analysis. In this extraordinarily rich, deep, and wide ranging life in research, all the strands are woven together. By investing that wealth into building and leading the Foundations of Software Engineering group at Microsoft Research (since August 1998), Gurevich professes his conviction that there is nothing more practical than a good theory.

References

1. E. Börger, *High level system design and analysis using Abstract State Machines.* In: Hutter, D., Stephan, W., Traverso, P., Ullmann, M. (eds): Current Trends in Applied Formal Methods (FM-Trends 98). LNCS 1641, pp. 1-43. Springer-Verlag, Berlin etc. (1999)
2. E. Börger, E. Grädel, and Y. Gurevich, *The Classical Decision Problem.* Perspectives in Mathematical Logic, Springer-Verlag Berlin etc. (1997), pp. XII+482.



Yuri Gurevich

Table of Contents

Invited Papers

Contributed Papers

Background, Reserve, and Gandy Machines

Andreas Blass[1,⋆] and Yuri Gurevich[2]

[1] Mathematics Dept., University of Michigan, Ann Arbor, MI 48109–1109, U.S.A.
ablass@umich.edu
[2] Microsoft Research, One Microsoft Way, Redmond, WA 98052, USA
gurevich@microsoft.com

Abstract. The reserve of a state of an abstract state machine was defined to be a "naked set". In applications, it may be convenient to have tuples, sets, lists, arrays, etc. defined ahead of time on all elements, including the reserve elements. We generalize the notion of reserve appropriately. As an application, we solve a foundational problem in Gandy's formalization of mechanical devices.

Part 1
Introduction and Preliminaries

1 Introduction

In this paper, we address two closely related foundational issues. We encountered these issues in connection with the notion of "reserve" in abstract state machines (ASMs), but they are of somewhat broader relevance, for example to the computing mechanisms described in [Gandy 1980]. In this introduction, we shall discuss these issues in general terms, specializing later to the cases of primary interest.

Algorithms often need to increase their working space, and there are two ways to view this increase. One is that the additional space was really there all along but was not previously used; the other is that genuinely new space is created. For example, from the first viewpoint, a Turing machine has an infinite tape, only finitely much of which is in use at any stage of the computation. From the second viewpoint, a Turing machine's tape is finite but new squares can be appended to it as needed.

For foundational purposes, it is usually more convenient to adopt the first viewpoint, so as not to have to worry about the nature of newly created elements. In

⋆ Preparation of this paper was partially supported by a grant from Microsoft Corporation.

P. Clote and H. Schwichtenberg (Eds.): CSL 2000, LNCS 1862, pp. 1–17, 2000.

particular, ASMs have, by definition [Gurevich 1995], an infinite reserve which is a part of the base set. All the basic functions and relations except equality take only their default values if at least one of the arguments belongs to the reserve, and no basic function outputs a reserve element. When a new element is needed in the active part of the state, one is imported from the reserve.

Although the reserve has no internal structure (except equality), it is often desirable to have some external structure over it. For example, in [BGS 1999] every state was required to include all the hereditarily finite sets over its atoms. This means that finite sets of reserve elements (and finite sets of these, etc.) were present, with their membership relation. Thus, when a reserve element is imported, sets involving it already exist and do not need to be created separately (by importing additional elements and appropriately defining the membership relation on them). Similarly, one might want to have other sorts of structure already available, for example lists or arrays of reserve elements.

The first issue treated in this paper is to make precise the notion of a sort of structure (like sets, or lists, or arrays) that can exist above a set of atoms without putting any structure (except equality) on the atoms themselves. We formalize this in the notion of a background class of structures. Thus, for example, the background class relevant to [BGS 1999] would consist of structures of the form: set U of atoms plus all hereditarily finite sets over U (as well as isomorphic copies of such structures). The idea is that such a class of structures specifies the constructions (like finite sets) available as "background" for algorithms.

The second issue is the choice of elements to be imported from the reserve. If the importation is to be algorithmic, it must be non-deterministic, since an algorithm has no way to distinguish one reserve element from another. But this sort of non-determinism is intuitively much more benign than general non-determinism. We attempt to capture what accounts for this intuition, by introducing the notion of inessential non-determinism. The idea here is that the various options allowed by the non-determinism all lead to isomorphic states, so that it makes no difference which option is chosen.

Alternatively, one could insist on determinism, specifying a particular one of the available reserve elements to be imported. This is the approach used in [Gandy 1980]. The price of this insistence is that the specification cannot be algorithmic or even canonical. We shall show how to turn a Gandy-style deterministic, non-algorithmic process into a non-deterministic algorithm of the sort described above, and we shall prove that Gandy's notion of "structural" for his processes corresponds to our notion of "inessential non-determinism."

2 Structures

The notion of (first-order) structure is found in textbooks on mathematical logic; for example see [Enderton 1972]. We use a slight modification of the notion of classical structures [Gurevich 1991].

2.1 Syntax

A *vocabulary* is a finite collection of function names, each of a fixed arity. Some function names may be marked as *relational*. Every vocabulary contains the equality sign, the nullary names `true`, `false`, `undef`, the unary name `Boole`, and the names of the usual Boolean operations. With the exception of `undef`, all these *logic names* are relational. A function name can be marked (by the vocabulary) as *static*.

Terms (more exactly *ground terms*; by default, terms are ground in this article) are defined by the usual induction. A nullary function name is a term. If f is a function name of positive arity j and if $t_1, \ldots, t_j$ are terms, then $f(t_1, \ldots, t_j)$ is a term. If the outermost function name is relational, then the term is *Boolean*.

2.2 Semantics

A *structure* X of vocabulary Υ is a nonempty set S (the *base set* of X) together with interpretations of the function names in Υ over S. Elements of S are also called elements of X. A j-ary function name is interpreted as a function from S^j to S, a *basic function* of X. We identify a nullary function with its value. Thus, in the context of a given structure, `true` means a particular element, namely the interpretation of the name `true`; the same applies to `false` and `undef`. It is required that `true` be distinct from the interpretations of the names `false` and `undef`. The interpretation of a j-ary relation R is a function from S^j to $\{\texttt{true}, \texttt{false}\}$, a *basic relation* of X. The equality sign is interpreted as the identity relation on the base set. Think about a basic relation R as the set of tuples $\bar{a}$ such that $R(\bar{a}) = \texttt{true}$. If relation R is unary it can be viewed as a *universe*. `Boole` is (interpreted as) the universe $\{\texttt{true}, \texttt{false}\}$. The Boolean operations behave in the usual way on `Boole` and produce `false` if at least one of the arguments is not Boolean. `undef` allows us to represent intuitively-partial functions as total.

The *domain* of a non-relational j-ary basic function f is the set of j-tuples $\bar{a}$ such that $f(\bar{a}) \neq \texttt{undef}$. The *range* of f is the set of elements $f(\bar{a})$ different from `undef` where $\bar{a}$ ranges over all j-tuples of elements of the base set.

A straightforward induction gives the value $Val(t, X)$ of a term t in a structure X whose vocabulary includes that of t. If $Val(t, X) = Val(t', X)$, we may say that $t = t'$ in X. If $t = \texttt{true}$ (resp. $t = \texttt{false}$) in X, we may say that t holds or is true (resp. fails or is false) in X.

3 Sequential-Time and Abstract-State Postulates

Restrict attention to one-thread algorithms. (In the terminology of [Gurevich 1995], this means that we allow parallel algorithms but not independent agents.) Following [Gurevich 2000], we make the following assumptions.

Sequential Time

Postulate 1 (Deterministic Sequential Time) Every deterministic algorithm A is associated with

- a set $\mathcal{S}(A)$ whose elements will be called *states* of A,
- a subset $\mathcal{I}(A)$ of $\mathcal{S}(A)$ whose elements will be called *initial states* of A, and
- a map $\tau_A : \mathcal{S}(A) \longrightarrow \mathcal{S}(A)$ that will be called the *one-step transformation* of A.

Postulate 2 (Nondeterministic Sequential Time) Every nondeterministic algorithm A is associated with

- a set $\mathcal{S}(A)$ whose elements will be called *states* of A,
- a subset $\mathcal{I}(A)$ of $\mathcal{S}(A)$ whose elements will be called *initial states* of A, and
- a relation $\tau_A \subseteq \mathcal{S}(A) \times \mathcal{S}(A)$ that will be called the *one-step transformation* of A.

Abstract State

Postulate 3 (Deterministic Abstract State) Let A be an arbitrary deterministic algorithm.

- States of A are first-order structures.
- All states of A have the same vocabulary.
- The one-step transformation τ_A does not change the base set of any state. Nor does it change the interpretations of static basic functions.
- $\mathcal{S}(A)$ and $\mathcal{I}(A)$ are closed under isomorphisms. Further, any isomorphism from a state X onto a state Y is also an isomorphism from $\tau_A(X)$ onto $\tau_A(Y)$.

Postulate 4 (Nondeterministic Abstract State) Let A be an arbitrary nondeterministic algorithm.

- States of A are first-order structures.
- All states of A have the same vocabulary.
- If $(X, X') \in \tau_A$ then X and X' have the same base set and the same basic static functions.
- $\mathcal{S}(A)$ and $\mathcal{I}(A)$ are closed under isomorphisms. Further, let ζ be an isomorphism from a state X onto a state Y. For every state X' with $(X, X') \in \tau_A$, there is a state Y' with $(Y, Y') \in \tau_A$ such that ζ is an isomorphism from X' onto Y'.

Notation. $\tau[A] \rightleftharpoons \tau_A$. Notation $\tau[A]$ is more convenient when the algorithm name is complex.

Lemma 3.1 (Symmetry Preservation) *Suppose that A is a deterministic algorithm and let $X \in \mathcal{S}(A)$. Every automorphism of X is also an automorphism of $\tau_A(X)$.*

Proof This is the last part of the deterministic abstract-state postulate applied to the special case that $X = Y$. □

3.1 Hereditarily Finite Sets

We recall a couple of set-theoretic notions.

A set x is *transitive* if it contains all elements of its elements: if $z \in y \in x$ then $z \in x$. The *transitive closure* $\mathrm{TC}(x)$ of a set x is the least transitive set that includes x. A set x is *hereditarily finite* if $\mathrm{TC}(x)$ is finite.

A set x may contain elements which are not sets; these are called *atoms*. The collection of atoms in $\mathrm{TC}(x)$ is the *atomic support* of x. Following Gandy [1980], the atomic support of a set x will be denoted $\mathrm{Sup}(x)$.

Let U be a set of atoms. The collection $\mathrm{HF}(U)$ of *hereditarily finite sets over U* is the collection of all hereditarily finite sets x with $\mathrm{Sup}(x) \subseteq U$. It is easy to see that $\mathrm{HF}(U)$ is the least set S such that every finite subset of $U \cup S$ is an element of S.

Corollary 3.2 *Consider a family $\{U_i : i \in I\}$ of subsets of U. We have*

$$\bigcap_i HF(U_i) = HF\Big(\bigcap_i U_i\Big)$$

Proof By definition of HF and of intersection, a set x belongs to $\bigcap_i \mathrm{HF}(U_i)$ if and only if $\mathrm{Sup}(x) \subseteq U_i$ for each i. But this is the same as saying $\mathrm{Sup}(x) \subseteq \bigcap_i U_i$, which is the definition of x belonging to $\mathrm{HF}\Big(\bigcap_i U_i\Big)$. □

Part 2
Background Classes and Reserve

4 Background Classes

4.1 Preliminaries

In this section, every vocabulary contains a unary predicate Atomic. We call this predicate and the logical symbols *obligatory* and all other symbols *non-obligatory*. If $X \models \mathrm{Atomic}(x)$, call x an *atom* of X. The set of atoms of X will

be denoted Atoms(X). X is *explicitly atom-generated* if the smallest substructure of X that includes all atoms is X itself.

Given two structures X, Y of the same vocabulary, we write $X \leq Y$ to indicate that X is a substructure of Y. If $X \leq Y$ and X belongs to a class K of structures then X is a *K-substructure* of Y.

4.2 Main Definitions

Definition 4.1 A class K of structures of a fixed vocabulary is a *background class* if the following requirements BC0–BC3 are satisfied.

BC0 K is closed under isomorphisms.
BC1 For every set U, there is a structure $X \in K$ with Atoms(X) $= U$.
BC2 For all $X, Y \in K$ and every embedding (of sets) ζ : Atoms(X) $\to$ Atoms(Y), there is a unique embedding (of structures) η of X into Y that extends ζ.
BC3 For all $X \in K$ and every $x \in$ Base(X), there is a smallest K-substructure Y of X that contains x.

Definition 4.2 Suppose that K is a background class, $X \in K$, $S \subseteq$ Base(X), and F is the set of substructures $Y \leq X$ such that Y belongs to K and includes S. If F has a smallest member Y then Y is called the *envelope* $E_X(S)$ of S in X and Atoms(Y) is called the *support* $\mathrm{Sup}_X(S)$ of S in X.

Notice that the smallest background substructure of X that contains a particular element $x \in X$ is $E_X(\{x\})$. It is tempting to simplify the notation by writing simply $E_X(x)$, but this can lead to ambiguity if x is both an element and a subset of X.

BC3 asserts that, in a background structure, every singleton subset has an envelope.

Definition 4.3 *A background class K is* finitary *if, in every background structure, the support of every singleton set is finite.*

4.3 Analysis

Let K be a background class. Members of K are *background structures*, K-substructures are *background substructures*.

Lemma 4.4 *In BC2, if ζ is onto then η is onto as well.*

Proof Suppose that ζ is onto. By BC2 (existence), ζ^{-1} extends to an embedding $\theta : Y \to X$. The identity map $\zeta \circ \zeta^{-1}$: Atoms(Y) $\to$ Atoms(Y) extends to $\eta \circ \theta : Y \to Y$. By BC2 (uniqueness), $\eta \circ \theta$ is the identity map on Y. It follows that η is onto. □

Lemma 4.5 *Suppose Z is a background structure, X, Y are background substructures of Z, $U = Atoms(X)$ and $V = Atoms(Y)$.*

1. *If $U \subseteq V$ then the identity on X is the unique embedding of X into Y that is the identity on U.*
2. *If $U \subseteq V$ then $X \leq Y$.*

Proof

1. Suppose that $U \subseteq V$ and let ζ be the identity on U and thus an embedding of U into V. By BC2 (existence), there is an extension of ζ to an embedding η of X into Y and therefore into Z. The identity θ on X is another extension of ζ that is an embedding of X into Z. By BC2 (uniqueness), $\eta = \theta$.

2. follows from 1. □

Lemma 4.6 *In a background structure X, every set U of atoms has an envelope.*

Proof By BC1, there is a background structure Y with $\text{Atoms}(Y) = U$. Let ζ be the identity map on U. By BC2 (existence), ζ extends to an embedding $\eta : Y \to X$. Let Z be the range of η. Clearly, $\text{Atoms}(Z) = U$. By BC0, Z is a background structure and thus a background substructure of X.

By Lemma 4.5, Z is included in every background substructure of X that includes U. This means that $Z = E(U)$. □

It follows that $\{a\}$ has an envelope for every atom a. This is weaker than BC3 which asserts that, in a background structure, every singleton subset has an envelope. Until now we used only BC0–BC2. BC3 does not follow from BC0–BC2, as the following example shows.

Example 4.7 Let K be the class of structures X satisfying the following conditions. The logic elements `true`, `false`, and `undef` are distinct. If $\text{Atoms}(X)$ is empty then X contains no non-logic elements. Otherwise the non-logic part of X consists of atoms and exactly one non-atomic element. It is easy to see that this class K satisfies BC0–BC2. However, if X has more than one atom and x is the unique non-logic non-atomic element, then $\{x\}$ does not have an envelope.

Lemma 4.8 *Every background class has the following property.*

BC3′ *In a background structure X, every $S \subseteq Base(X)$ has an envelope.*

Proof Let $U \rightleftharpoons \bigcup\{\mathrm{Sup}(\{x\}) : x \in S\}$. By Lemma 4.6, U has an envelope $E(U)$. We show that $E(U)$ is also the envelope of S.

$S \subseteq E(U)$. Indeed, for all $x \in S$, $\mathrm{Atoms}(E(\{x\})) = \mathrm{Sup}(\{x\}) \subseteq U$ so that, by Lemma 4.5, $E(\{x\}) \leq E(U)$.

$E(U)$ is the smallest K-substructure of X that includes S. Indeed, let Z be any K-substructure of X that includes S. For every $x \in S$, Z includes $E(\{x\})$ and therefore includes $\mathrm{Sup}(\{x\}) = \mathrm{Atoms}(E(\{x\}))$. Hence Z includes U. Hence Z includes $E(U)$. □

Lemma 4.9 *Every background class has the following property.*

BC3″ *For all $X \in K$, the intersection of any family of K-substructures of X is a K-substructure of X.*

Proof Let F be a family of K-substructures of X. We prove that the substructure $\bigcap F$ is a background structure. Let $U = \bigcap\{\mathrm{Atoms}(Y) : Y \in F\}$. It suffices to prove that $\bigcap F = E(U)$.

$E(U) \leq \bigcap F$. Indeed, by the definition of $E(U)$, $E(U)) \leq Y$ for all $Y \in F$.

$\bigcap F \leq E(U)$. Indeed let x be an element of $\bigcap F$. Every $Y \in F$ contains x, therefore includes $E(\{x\})$, and therefore includes $\mathrm{Atoms}(E(\{x\}))$. It follows that $\mathrm{Atoms}(E(\{x\})) \subseteq U$. By Lemma 4.5, $E(\{x\}) \leq E(U)$ and therefore $E(U)$ contains x. □

This proof gives rise to the following corollary.

Corollary 4.10 *Assume that X is a background structure and let $U_i \subseteq \mathrm{Atoms}(X)$ for all $i \in I$. Then*

$$\bigcap_i E(U_i) = E(\bigcap_i U_i)$$

Lemma 4.11 *In Definition 4.1, BC3 can be replaced with BC3″.*

Proof Assume BC3″ and let $X \in K$. Given an element x of X, let F be the collection of K-substructures Y of X such that Y contains x. By BC3″, $\bigcap F$ is a K-substructure of X. Clearly, it is the smallest K-substructure of X that contains x. □

Remark 4.12 The definition of background classes has a simple category-theory version. Consider two categories:

Υ-Str The category of structures for a vocabulary Υ, with embeddings as morphisms.

Set The category of sets, also with embeddings (i.e., one to one maps) as morphisms.

Of course Set is just the special case of Υ-Str, where the vocabulary Υ is empty. But we're interested in the case where Υ contains at least the unary predicate symbol Atomic. This symbol gives rise to a functor $\mathcal{F}$ from Υ-Str to Set. The functor sends each Υ-structure to its set of atoms; on morphisms it acts by restriction. Now a background class is a full subcategory C of Υ-Str that is closed under isomorphisms and under intersections and such that the functor $\mathcal{F}$ when restricted to C is an equivalence of categories C and Set.

Here "full subcategory" means a subclass of the objects, with all of the Υ-Str morphisms between them. And "closed under intersections" should be taken in the category-theoretic sense. In set-theoretic terminology, this means that, given a structure in C and some substructures, also in C, then their intersection should also be in C. One needs the single superstructure to make sense of the intersection.

Lemma 4.13 *Suppose that X is a background structure. For every permutation π of Atoms(X) there is a unique extension of π to an automorphism of X.*

Proof Use BC2 and Lemma 4.4. □

5 Examples of Background Classes

In this section we shall describe some specific background classes. Some of these were the motivation for the general definition of background class.

Recall that the non-obligatory part of a vocabulary is the part that is obtained by removing all logic names as well as the name Atomic.

5.1 Set Background

Up to isomorphism, the non-logic part of a background structure X with atoms U consists of the hereditarily finite sets over U. (See Part 1 about the hereditarily finite sets.) The only non-obligatory basic function of X is the containment relation $\in$. For future reference, this background class will be called the set-background class SB.

There are other versions of set backgrounds. The vocabulary of SB can be enriched in various ways. We consider two ways to do that.

1. The additional basic functions are $\emptyset$, Singleton$(x) = \{x\}$, and

$$\text{BinaryUnion}(x, y) =\rightleftharpoons \begin{cases} x \cup y & \text{if both } x \text{ and } y \text{ are sets} \\ \emptyset & \text{otherwise} \end{cases}$$

The resulting background class is explicitly atom-generated.

2. The additional basic functions are as in [BGS 1999]: $\emptyset$, $\mathrm{Pair}(x, y) \rightleftharpoons \{x, y\}$ and

$$\mathrm{UnaryUnion}(x) \rightleftharpoons \begin{cases} \bigcup_{y \in x} y & \text{if } x \text{ is a set} \\ \emptyset & \text{otherwise} \end{cases}$$

$$\mathrm{TheUnique}(x) \rightleftharpoons \begin{cases} a & \text{if } x \text{ is a singleton } \{a\} \\ \emptyset & \text{otherwise} \end{cases}$$

Again the resulting background class is explicitly atom-generated.

5.2 String Background

The set of non-logic elements of a background structure with atoms U is the set of strings of elements of U. Let the vocabulary contain symbols for the nullary function Nil (the empty string), the unary function sending a member of U to the one-term string containing it, and the binary operation of concatenation of strings. Then this background class is explicitly atom-generated. If desired, one can introduce additional basic functions $\mathrm{Head}(x)$ and $\mathrm{Tail}(x)$ defined by induction on x. If x is the empty string Nil then $\mathrm{Head}(x) \rightleftharpoons \mathrm{Tail}(x) \rightleftharpoons \mathrm{Nil}$. If $x = ay$ where a is an atom, then $\mathrm{Head}(x) \rightleftharpoons a$ and $\mathrm{Tail}(x) \rightleftharpoons y$.

5.3 List Background

Up to isomorphism, the non-logic part of a backgound structure X with atoms U consist of the lists over U. The terms in the lists can be elements of U or other lists. (So lists differ from the strings in the preceding example in that nesting is allowed.) The non-logic basic functions are Nil and Append. Nil desginates the empty list. Given an atom or a list x and given a list $\langle y_1, \ldots, y_n \rangle$, $\mathrm{Append}(x, y) \rightleftharpoons \langle x, y_1, \ldots, y_n \rangle$. Every list has a unique presentation. As above, this allows us to introduce additional basic functions $\mathrm{Head}(x)$ and $\mathrm{Tail}(x)$ where x ranges over lists. In either version, this background class is explicitly atom-generated.

5.4 Set/List Background

Up to isomorphism, the set of non-logic elements of a background structure with atoms U is the least set V such that

- $U \subseteq V$,
- for every natural number n, if $x_1, \ldots, x_n \in V$ then $\{x_1, \ldots, x_n\} \in V$ and $\langle x_1, \ldots, x_n \rangle \in V$.

Here we do not adopt any of the codings of lists as sets; we regards sets and lists as independent basic constructions.

We leave the choice of the vocabulary to the reader.

5.5 A Non-finitary Background

All example background classes above are finitary. To obtain a non-finitary background class, modify the string-background class by allowing infinite strings.

6 Background Structures and the Reserve

Fix a background class BC. Call the vocabulary Υ_0 of BC the *background vocabulary*, function names in Υ_0 the *background function names*, and members of BC *background structures*.

Definition 6.1 Let A be an algorithm, deterministic or nondeterministic. BC is the *background* of A if the following conditions are satisfied.

• The vocabulary Υ of A includes the background vocabulary Υ_0, and every background function name is static in Υ.

• For every state X of A, the Υ_0-reduct of X (obtained by "forgetting" the basic functions with names in $\Upsilon - \Upsilon_0$) is a background structure. □

Fix a (deterministic or nondeterministic) algorithm A with background BC, and let Υ be the vocabulary of A. The basic functions of A with names in Υ_0 will be called the *background basic functions* of A; the other basic functions of A will be called the *foreground basic functions* of A.

Definition 6.2 Let X be a state of A. An element $x \in \text{Base}(X)$ is *exposed* if x belongs to the range of a foreground function or else x occurs in a tuple that belongs to the domain of a foreground function. □

Recall the property BC3′ of background classes: for every background structure X, every subset of $\text{Base}(X)$ has an envelope in X.

Definition 6.3 The *active part* of a state X of the algorithm A is the envelope of the set of exposed elements; we denote it by $\text{Active}(X)$. The *reserve* of X is the set of atoms of X that do not belong to the active part.

Lemma 6.4 *Every permutation of the reserve of X gives rise to a unique automorphism of X that is the identity on the active part of X.*

Proof Let π be a permutation of the reserve of X. Set $\pi(a) \rightleftharpoons a$ for all atoms a in the active part of X; the extended permutation will be also called π. By Lemma 4.13, there is a unique automorphism θ of the Υ_0-reduct of X that extends π. By definition of active part, any such automorphism that is the identity on the active part is necessarily an automorphism of the full Υ-structure X. □

We remark for future reference that any isomorphism $X \cong Y$ between states of A maps $\text{Active}(X)$ isomorphically onto $\text{Active}(Y)$.

7 Inessential Nondeterminism

The symmetry preservation Lemma 3.1 inspires the following definition.

Definition 7.1 Suppose that A is a nondeterministic algorithm with background BC. A is *essentially deterministic* (or *inessentially nondeterministic*) if the following holds for all states X of A. If (X, X') and (X, X'') belong to τ_A then there is an isomorphism from X' onto X'' that coincides with the identity on Active(X).

Corollary 7.2 *Suppose that A is an inessentially nondeterministic algorithm with background BC. Let $(X, X') \in \tau_A$, $(Y, Y') \in \tau_A$, ζ be an isomorphism from X onto Y, and ζ_0 be the restriction of ζ to Active(X). Then ζ_0 extends to an isomorphism from X' onto Y'.*

Proof By Postulate 4 (Nondeterministic Abstract State), ζ is an isomorphism from X' to some state Y'' with $(Y, Y'') \in \tau_A$. Since A is inessentially nondeterministic, there is an isomorphism $\theta : Y'' \cong Y$ that coincides with the identity on Active(Y), which equals $\zeta($Active$(X))$ by the remark at the end of the last section. Then $\theta \circ \zeta$ is an isomorphism from X' to Y' and agrees with ζ on Active(X), i.e., it extends ζ_0. □

Part 3
Nondeterministic Choice Problem for Gandy Machines

8 Gandy Machines

Following Gandy [1980], fix an infinite countable set U of atoms. Recall that HF(U) is the collection of hereditarily finite sets over U (see Part 1). Let $\mathcal{G}$ be the structure

$$(U \cup \mathrm{HF}(U), \in, U)$$

Every permutation π of U naturally extends to an automorphism of $\mathcal{G}$ as follows: if $x \in \mathrm{HF}(U)$ then $\pi x \rightleftharpoons \{\pi y : y \in x\}$. It is easy to see that every automorphism of $\mathcal{G}$ is obtained this way.

A subset S of HF(U) is *structural* if it is closed under automorphisms of $\mathcal{G}$. In other words, S is structural if and only if, for every $x \in S$ and for every permutation π of U, we have $\pi x \in S$. The following definition plays an important role in this part.

Definition 8.1 Let S be a structural subset of $\mathrm{HF}(U)$. A function $F : S \to \mathrm{HF}(U)$ is *structural* if, for every $x \in S$ and for every permutation π of U, there is a permutation ρ of U that pointwise fixes $\mathrm{Sup}(\{\pi x\})$ and such that $\rho\pi Fx = F\pi x$.

Gandy defined only structural functions over $\mathrm{HF}(U)$ but used structural functions over arbitrary structural subsets of $\mathrm{HF}(U)$. The following lemma clarifies the issue of structural functions over $\mathrm{HF}(U)$ vs. structural functions over structural subsets of $\mathrm{HF}(U)$.

Lemma 8.2

1. A structural function F over a structural set S extends to a structural function over HF(U).

2. Suppose that F is a structural function over HF(U) and let S be a structural subset of HF(U). Then the restriction $F|S$ of F to S is a structural function over S.

Proof

1. Set $F(x) \rightleftharpoons \emptyset$ for all $x \in \mathrm{HF}(U) - S$. We show that the extended function F is structural over $\mathrm{HF}(U)$. Let $x \in \mathrm{HF}(U)$ and π be an arbitrary permutation of U. We need to prove the existence of an appropriate permutation ρ. If $x \in S$, then the existence of an appropriate ρ follows from the structurality of F over S. Suppose that $x \notin S$. By the structurality of S, $\pi x \notin S$. Then $\emptyset = Fx = F\pi x = \pi Fx$. The desired ρ is the identity permutation of U.

2. Let $x \in S$ and π be an arbitrary permutation of U. Since F is structural over $\mathrm{HF}(U)$, there is a permutation ρ of U that pointwise fixes $\mathrm{Sup}(\{\pi x\})$ and such that $\rho\pi Fx = F\pi x$. If S contains x then it contains πx as well because S is structural. It follows that $\rho\pi(F|S)x = (F|S)\pi x$ for all $x \in S$. □

Now we are ready to recall the notion of Gandy machine at the level of detail appropriate for this paper.

Definition 8.3 A *Gandy machine* M is a pair (S, F)

- S is a structural subset of $\mathrm{HF}(U)$, and
- F is a structural function from S into S, and
- some additional constraints are satisfied.

Intuitively, S is the set of states of M and F is the one-step transition function. The additional constraints are not important for our purposes in this paper.

9 The Nondeterministic Choice Problem

We start with an example Gandy machine $M^0 = (S^0, F^0)$. S^0 is the collection of finite subsets of U. Obviously, S^0 is structural. If $x \in S^0$, then $F(x) \rightleftharpoons x \cup \{a\}$ where a is an atom in $U - x$. We check that F^0 is structural as well. Suppose that $x = \{a_1, \ldots, a_n\} \in S^0$ where $a_1, \ldots, a_n$ are distinct. Then $F^0 x = \{a_1, \ldots, a_n, b\}$ for some atom $b \notin x$ so that $a_1, \ldots, a_n, b$ are all distinct. Let π be a permutation of U. Then

$$\begin{aligned} \pi x &= \{\pi a_1, \ldots, \pi a_n\} \\ \pi F^0 x &= \{\pi a_1, \ldots, \pi a_n, \pi b\} \quad \text{where } \pi b \notin \pi x \\ F^0 \pi x &= \{\pi a_1, \ldots, \pi a_n, c\} \quad \text{for some } c \notin \pi x \end{aligned}$$

The desired ρ transposes πb and c and leaves other atoms intact.

Thus, M^0 satisfies the part of the definition of Gandy machine that we gave explicitly; the reader familiar with [Gandy 1980] is invited to check that the "additional constraints" that we alluded to are also satisfied.

Now consider an arbitrary Gandy machine $M = (S, F)$ and let $x \in S$. Think about x as the current state of M, so that Fx is the next state of M. It is possible that $\mathrm{Sup}(\{Fx\})$ contains atoms that are not in $\mathrm{Sup}(\{x\})$; that certainly happens in the case of our example machine M^0. The choice of such new atoms should not matter. That is why Gandy requires that F is structural. Two questions arise.

1. Is the structurality of F a correct requirement? That is, does it capture the idea that the choice of new elements is irrelevant?
2. Is there a better solution of this nondeterministic choice problem?

We believe that the answer to the second question is positive. To this end we will propose a nondeterministic formalization of Gandy machines. The answer to the first question is positive as well if one sticks to deterministic machines; we will show that the structurality requirement is equivalent to the intuitively expected requirement that the nondeterministic versions of Gandy machines are essentially deterministic in the sense of Section 7.

10 Nondeterministic (Specifications for) Gandy Machines

Let S be a structural subset of $\mathrm{HF}(U)$ and $F : S \to S$ be any unary operation over S. Think about $M = (S, F)$ as a machine, like a Gandy machine. Of course, we are primarily interested in the case when M is a Gandy machine, but for technical reasons we consider a more general case. We define a nondeterministic algorithm A, or A_M, that may serve as a nondeterministic specification for M.

Definition 10.1 The *nondeterministic specification* of $M = (S, F)$ is the algorithm A defined as follows.

All states of A have the same base set, namely the set $U \cup \mathrm{HF}(U)$ extended with three additional elements (interpreting) $\mathtt{true}, \mathtt{false}, \mathtt{undef}$. A has only three non-logic basic functions. Two of them are static: the unary relation Atomic and the containment relation $x \in y$. The third basic function is a nullary dynamic function Core that gives the current state of M. For brevity, let $\mathrm{Core}(X)$ be the value of Core at state X and let $\mathrm{Sup}(X) \rightleftharpoons \mathrm{Sup}(\{\mathrm{Core}(X)\})$.

The one-step transition relation τ_A consists of pairs (X, Y) of states of A such that $\mathrm{Core}(Y) = F(\mathrm{Core}(X))$ or, more generally, there is a permutation π of U that pointwise fixes $\mathrm{Sup}(X)$ and such that $\mathrm{Core}(Y) = \pi F(\mathrm{Core}(X)))$.

To explain this definition, consider the example M^0 from the preceding section. Abbreviate A_{M^0} to A^0. Abbreviate τ_{A^0} to τ^0. In this situation, $\mathrm{Sup}(X) = \mathrm{Core}(X)$ for all states X of S. Fix a particular state X of A^0. What are the states Y such that $(X, Y) \in \tau^0$? It is easy to see that these are exactly the state Y such that $\mathrm{Sup}(Y)$ consists of the atoms in $\mathrm{Sup}(X)$ plus one additional atom a. Any atom in $U - \mathrm{Sup}(X)$ will do. No atom in $U - \mathrm{Sup}(X)$ has preferential treatment or is discriminated against.

Remark 10.2 Gandy does not specify initial states of his machines. Accordingly we ignore initial states as well.

It is easy to see that A is a nondeterministic algorithm with background SB described in Subsection 5.1. The only exposed element of a state X of A is $\mathrm{Core}(X)$. Accordingly the active part $\mathrm{Active}(X)$ of X is $\mathrm{Sup}(X) \cup \mathrm{HF}(\mathrm{Sup}(X))$ plus the elements $\mathtt{true}, \mathtt{false}, \mathtt{undef}$. Hence $\mathrm{Reserve}(X) = U - \mathrm{Sup}(X)$.

We saw that every permutation π of U extends to an automorphism of the structure $\mathcal{G} = (U \cup \mathrm{HF}(U), \in, U)$. However, π is not necessarily an automorphism of X because it can move $\mathrm{Core}(X)$. It is an automorphism of X if and only if $\pi(\mathrm{Core}(X)) = \mathrm{Core}(X)$. Identify a permutation π of $\mathrm{Reserve}(X)$ with the permutation of U which pointwise fixes $\mathrm{Sup}(X)$ and coincides with π on $\mathrm{Reserve}(X)$.

Corollary 10.3 *Let (X, Y) be states of A. Then $(X, Y) \in \tau_A$ if and only if there is an permutation π of $\mathrm{Reserve}(X)$ such that $\mathrm{Core}(Y) = \pi F(\mathrm{Core}(X))$.*

11 Essential Determinism and Structurality

Let S be a structural subset of $\mathrm{HF}(U)$ and F be any unary operation over S. Further let A be the nondeterministic specification for (S, F). Abbreviate τ_A to τ.

Theorem 11.1 *The following are equivalent.*

1. *A is essentially deterministic.*
2. *F is structural over S.*

Proof First we assume 1 and prove 2. Let $x \in S$ and let π be any permutation of U. We construct the desired ρ.

Let X be the state of A with $\mathrm{Core}(X) = x$ and let πX be the state of A with $\mathrm{Core}(\pi X) = \pi x$. Since S is structural, it contains πx and thus πX is a legitimate state of A.

View π as an isomorphism from X to πX. Since A is essentially deterministic, there is an isomorphism η from $\tau(X)$ to $\tau(\pi X)$ which coincides with π on $\mathrm{Active}(X)$ and in particular on $\mathrm{Sup}(\{x\})$. The desired $\rho z \rightleftharpoons \eta\pi^{-1}z$ for all $z \in \mathrm{HF}(U)$.

If $\pi a \in \mathrm{Sup}(\{\pi x\})$ then $\rho(\pi a) = \eta\pi^{-1}\pi a = \eta a = \pi a$. Thus ρ is the identity on $\mathrm{Sup}(\{\pi x\})$.

Since η is an isomorphism from τX onto $\tau(\pi X)$, we have $\eta\mathrm{Core}(\tau X) = \mathrm{Core}(\tau\pi X)$, that is $\eta Fx = F\pi x$. Therefore $\rho\pi Fx = \eta\pi^{-1}\pi Fx = \eta Fx = F\pi x$.

Second we assume 2 and prove 1. Suppose that (X, Y) and (X, Z) belong to τ. We need to prove that there is an isomorphism from Y onto Z that pointwise fixes $\mathrm{Active}(X)$.

Without loss of generality $\mathrm{Core}(Y) = F(\mathrm{Core}(X))$. Indeed, suppose that 2 is proved in this special case. Now consider the general case, and let X' be the state of A with $\mathrm{Core}(X') = F(\mathrm{Core}(X))$. By the special case of 2, there is an isomorphism ζ from X' onto Y that pointwise fixes $\mathrm{Active}(X)$. Similarly, there is an isomorphism η from X' onto Z that pointwise fixes $\mathrm{Active}(X)$. Then $\eta \circ \zeta^{-1}$ is an automorphism from Y onto Z that pointwise fixes $\mathrm{Active}(X)$.

Let $x \rightleftharpoons \mathrm{Core}(X), y \rightleftharpoons \mathrm{Core}(Y)$ and $z \rightleftharpoons \mathrm{Core}(Z)$. We have $y = Fx$. Since $(X, Z) \in \tau$, there exists a permutation π of U that pointwise fixes $\mathrm{Sup}(\{x\})$ and such that $z = \pi Fx$. Since F is structural, there exists a permutation ρ of U that pointwise fixes $\mathrm{Sup}(\{\pi x\})$ and such that $\rho\pi Fx = F\pi x$. Since π pointwise fixes $\mathrm{Sup}(\{x\})$, we have $\pi x = x$ so that ρ pointwise fixes $\mathrm{Sup}(\{x\})$ (and therefore pointwise fixes $\mathrm{Active}(X)$) and $\rho z = \rho\pi Fx = F\pi x = Fx = y$. Then ρ^{-1} pointwise fixes $\mathrm{Active}(X)$ and takes y to z. □

References

BGS 1999 Andreas Blass, Yuri Gurevich and Saharon Shelah, "Choiceless Polynomial Time", Annals of Pure and Applied Logic 100 (1999), 141–187.

Enderton 1972 Herbert B. Enderton, "A Mathematical Introduction to Logic," Academic Press.

Gandy 1980 Robin Gandy, "Church's thesis and principles for mechanisms" in: "The Kleene Symposium" (ed. J. Barwise et al.), North-Holland, 1980, 123–148.

Gurevich 1991 "Evolving Algebras: An Attempt to Discover Semantics", Bulletin of European Assoc. for Theor. Computer Science, no. 43, Feb. 1991, 264–284. A slightly revised version appeared in G. Rozenberg and A. Salomaa, editors, "Current Trends in Theoretical Computer Science", World Scientific, 1993, 266–292.

Gurevich 1995 Yuri Gurevich, "Evolving Algebra 1993: Lipari Guide", in "Specification and Validation Methods", Ed. E. Boerger, Oxford University Press, 1995, 9–36.

Gurevich 2000 Yuri Gurevich, "Sequential Abstract State Machines Capture Sequential Algorithms", ACM Transactions on Computational Logic 1, to appear.

Choiceless Polynomial Time Computation and the Zero-One Law

Andreas Blass[1,⋆] and Yuri Gurevich[2]

[1] Mathematics Dept., University of Michigan, Ann Arbor, MI 48109–1109, USA
ablass@umich.edu
[2] Microsoft Research, One Microsoft Way, Redmond, WA 98052, USA
gurevich@microsoft.com

1 Introduction

This paper is a sequel to [2], a commentary on [7], and an abridged version of a planned paper that will contain complete proofs of all the results presented here.

The BGS model of computation was defined in [2] with the intention of modeling computation with arbitrary finite relational structures as inputs, with essentially arbitrary data structures, with parallelism, but without arbitrary choices. In the absence of any resource bounds, the lack of arbitrary choices makes no difference, because an algorithm could take advantage of parallelism to produce all possible linear orderings of its input and then use each of these orderings to make whatever choices are needed. But if we require the total computation time (summed over all parallel subprocesses) to be polynomially bounded, then there isn't time to construct all the linear orderings, and so the inability to make arbitrary choices really matters.

In fact, it was shown that choiceless polynomial time $\tilde{\text{C}}$PTime, the complexity class defined by BGS programs subject to a polynomial time bound, does not contain the parity problem: Given a set, determine whether its cardinality is even. Several similar results were proved, all depending on symmetry considerations, i.e., on automorphisms of the input structure.

Subsequently, Shelah [7] proved a zero-one law for $\tilde{\text{C}}$PTime properties of graphs. We shall state this law and discuss its proof later in this paper. For now, let us just mention a crucial difference from the earlier results in [2]: Almost all finite graphs have no non-trivial automorphisms, so symmetry considerations cannot be applied to them. Shelah's proof therefore depends on a more subtle concept of partial symmetry, which we explain in Section 8 below.

Finding the proof in (an early version of) [7] difficult to follow, we worked out a presentation of the argument for the main case, which we hope will be helpful for others interested in Shelah's ideas. We also added some related results, indicating the need for certain aspects of the proof and clarifying some of the concepts involved in it. Unfortunately, this material is not yet fully written up.

⋆ Preparation of this paper was partially supported by a grant from Microsoft Corporation.

P. Clote and H. Schwichtenberg (Eds.): CSL 2000, LNCS 1862, pp. 18–40, 2000.

The part already written, however, exceeds the space available to us in the present volume. We therefore present here an abridged version of that paper and promise to make the complete version available soon.

For simplicity, we shall often deal only with input structures that are undirected, loopless graphs, i.e., sets equipped with a symmetric, irreflexive binary relation of adjacency. We also restrict our attention to the uniform probability model. That is, we define the probability of a property (assumed isomorphism-invariant) of n-vertex graphs by considering all graphs with vertex set $\{1, 2, \ldots, n\}$ to be equally probable. The *asymptotic probability* of a property of graphs is defined as the limit, as $n \to \infty$, of its probability among n-vertex graphs. In general, a zero-one law says that properties have asymptotic probability 0 or 1, but, as we shall see, some care is needed in formulating the zero-one law for $\tilde{\text{C}}$PTime.

All the results discussed in this paper can be routinely extended to other contexts, such as directed graphs, or sets with several relations, including relations of more than two arguments. It is also routine to replace the uniform probability measure by one where all potential edges have probability p, a constant other than $\frac{1}{2}$. We do not discuss these generalizations further, because they complicate the notation without contributing any new ideas.

2 The Zero-One Law

We start with a very brief description of the BGS model of computation, just adequate to formulate the zero-one law. In Section 3, we shall give more details about the model, in preparation for a description of its proof.

The BGS model, introduced in [2], is a version of the abstract state machine (ASM) paradigm [6]. The input to a computation is a finite relational structure I. A state of the computation is a structure whose domain is $\text{HF}(I)$, which consists of the domain of I together with all hereditarily finite sets over it; the structure has the relations of I, some set-theoretical apparatus (for example the membership relation $\in$), and some dynamic functions. The computation proceeds in stages, always modifying the dynamic functions in accordance with the program of the computation. The dynamic functions are initially constant with value $\emptyset$ and they change at only finitely many arguments at each step. So, although $\text{HF}(I)$ is infinite, only a finite part of it is involved in the computation at any stage. The computation ends when and if a specific dynamic 0-ary function Halt acquires the value `true`, and the result of the computation is then the value of another dynamic 0-ary function Output.

This model was used to define choiceless polynomial time $\tilde{\text{C}}$PTime by requiring a computation to take only polynomially many (relative to the size of the input structure I) steps and to have only polynomially many active elements. (Roughly speaking, an element of $\text{HF}(I)$ is active if it participates in the updating of some dynamic function at some stage.) Also, Output was restricted to have Boolean values, so the result of a computation could only be true, or false, or undecided. (The "undecided" situation arises if the computation exhausts the allowed number of steps or the allowed number of active elements without Halt

becoming true.) We shall use the name *polynomial time BGS program* to refer to a BGS program, with Boolean Output, together with polynomial bounds on the number of steps and the number of active elements.

Two classes $\mathcal{K}_0$ and $\mathcal{K}_1$ of graphs are $\tilde{\text{C}}$PTime-*separable* if there is a polynomial time BGS program Π such that, for all input structures from $\mathcal{K}_0$ (resp. $\mathcal{K}_1$), Π halts with output `false` (resp. `true`) without exceeding the polynomial bounds. It doesn't matter what Π does when the input is in neither $\mathcal{K}_0$ nor $\mathcal{K}_1$.

Theorem 1 (Shelah's Zero-One Law) *If $\mathcal{K}_0$ and $\mathcal{K}_1$ are $\tilde{C}$PTime-separable classes of undirected graphs, then at least one of $\mathcal{K}_0$ and $\mathcal{K}_1$ has asymptotic probability zero.*

An equivalent formulation of this is that, for any given polynomial time BGS program, either almost all graphs produce output true or undecided or else almost all graphs produce output false or undecided. It is tempting to assert the stronger claim that either almost all graphs produce true, or almost all produce false, or almost all produce undecided. Unfortunately, this stronger claim is false; a counterexample will be given after we review the definition of BGS programs in Section 3.

The theorem was, however, strengthened considerably in another direction in [7]. It turns out that the number of steps in a halting computation is almost independent of the input.

Theorem 2 *Let a BGS program Π with Boolean output and a polynomial bound for the number of active elements be given. There exist a number m, an output value v, and a class $\mathcal{C}$ of undirected graphs, such that $\mathcal{C}$ has asymptotic probability one and such that, for each input $I \in \mathcal{C}$, one of the two following alternatives holds. Either Π on input I halts after exactly m steps with output value v and without exceeding the given bound on active elements, or Π on input I exceeds the bound on active elements by step m.*

Notice that this theorem does not assume a polynomial bound on the number of steps. It is part of the conclusion that the number of steps is not only polynomially bounded but constant as long as the input is in $\mathcal{C}$ and the number of active elements obeys its bound.

Intuitively, bounding the number of active elements, without bounding the number of computation steps, amounts to a restriction on space, rather than time. Thus, Theorem 2 can be viewed as a zero-one law for choiceless polynomial space computation.

The class $\mathcal{C}$ in the theorem actually has a fairly simple description; it consists of the graphs that have at least n_1 nodes and satisfy the strong extension axioms to be defined in Section 7 below for up to n_2 variables. The parameters n_1 and n_2 in this definition can be easily computed when the program Π and the polynomial bound on the number of active elements are specified.

3 BGS Programs

In this section, we review the syntax and semantics of BGS programs, as well as the concept of active elements. These are the ingredients used in defining $\tilde{\text{C}}$PTime in [2].

We identify the truth values `false` and `true` with the sets $0 = \emptyset$ and $1 = \{0\}$, respectively. Thus, relations can be regarded as functions taking values in $\{0, 1\}$.

Definition 3 Our *function symbols* are

- the logical symbols, namely $=$ and the connectives $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$, `true`, and `false`,
- the set-theoretic function symbols $\in$, $\emptyset$, Atoms, $\bigcup$, TheUnique, and Pair,
- the input predicate symbol A, and
- finitely many dynamic function symbols.

The intended interpretation of $\bigcup x$, where x is a family of sets and atoms, is the union of the sets in x (ignoring the atoms). If x is a set with exactly one member then TheUnique(x) is that member. Pair(x, y) means $\{x, y\}$. The input predicate A denotes the adjacency relation of the input graph. The intended meanings (and arities) of the other symbols should be clear. We adopt the convention that if a function is applied to an inappropriate argument (like $\bigcup$ applied to an atom or A applied to sets) then the value is $\emptyset$.

The function symbols $\in$, A, and the logical symbols are called *predicates* because their intended values are only `true` and `false`.

In addition to function symbols, we use a countably infinite supply of variables and we use certain symbols introduced in the following definitions of terms and rules.

Definition 4 *Terms* and *Boolean terms* are defined recursively as follows.

- Every variable is a term.
- If f is a j-ary function symbol and $t_1, \ldots, t_j$ are terms, then $f(t_1, \ldots, t_j)$ is a term. It is Boolean if f is a predicate.
- If v is a variable, $t(v)$ a term, r a term in which v is not free, and $\varphi(v)$ a Boolean term, then
$$\{t(v) : v \in r : \varphi(v)\}$$
is a term.

The construction $\{t(v) : v \in r : \varphi(v)\}$ binds the variable v.

In connection with $\{t(v) : v \in r : \varphi(v)\}$, we remark that, by exhibiting the variable v in $t(v)$ and $\varphi(v)$, we do not mean to imply that v must actually occur there, nor do we mean that other variables cannot occur there. We are merely indicating the places where v could occur free. The "two-colon" notation $\{t(v) : v \in r : \varphi(v)\}$ is intended to be synonymous with the more familiar "one-colon" notation $\{t(v) : v \in r \wedge \varphi(v)\}$. By separating the $v \in r$ part from $\varphi(v)$, we indicate

the computational intention that the set should be built by running through all members of r, testing for each one whether it satisfies φ, and collecting the appropriate values of t. Thus $\{t(v) : v \in r : v \in r'\}$ and $\{t(v) : v \in r' : v \in r\}$ are the same set, but produced in different ways. (Such "implementation details" have no bearing on the results but provide useful intuitive background for some of our definitions.)

Definition 5 *Rules* are defined recursively as follows.

- `Skip` is a rule.
- If f is a dynamic j-ary function symbol and $t_0, t_1, \ldots, t_j$ are terms, then

$$f(t_1, \ldots, t_j) := t_0$$

  is a rule, called an *update rule*.
- If φ is a Boolean term and R_0 and R_1 are rules, then

$$\texttt{if } \varphi \texttt{ then } R_0 \texttt{ else } R_1 \texttt{ endif}$$

  is a rule, called a *conditional rule*.
- If v is a variable, r is a term in which v is not free, and $R(v)$ is a rule, then

$$\texttt{do forall } v \in r,\ R(v) \texttt{ enddo}$$

  is a rule, called a *parallel combination*.

The construct `do forall` $v \in r,\ R(v)$ `enddo` binds the variable v.

Convention 6 When the "else" part is `Skip`, we use `if` φ `then` R to abbreviate `if` φ `then` R `else Skip endif`. We use `do in parallel` R_0, R_1`enddo` as an abbreviation for

```
do forall v ∈ Pair(true, false)
   if v =true then R0 else R1
   endif
enddo
```

The `do in parallel` construct applied to more than two rules means an iteration of the binary `do in parallel` .

Definition 7 A *program* is a rule with no free variables.

Convention 8 By renaming bound variables if necessary, we assume that no variable occurs both bound and free, and no variable is bound twice, in any term or rule.

Throughout much of this paper, the context of our discussion will include a fixed program Π. In such situations, we adopt the following convention.

Convention 9 When we refer to a term or rule within Π, we mean a specific occurrence of the term or rule in Π.

Since a program Π has no free variables, every variable v occurring in it is bound exactly once, either by a term $\{t : v \in r : \varphi\}$ or by a rule `do forall` $v \in r$, R `enddo`.

Definition 10 If v is bound by $\{t : v \in r : \varphi\}$, then the *scope* of v consists of the exhibited occurrence of v as well as t and φ. If v is bound by `do forall` $v \in r$, R `enddo`, then the *scope* of v consists of its exhibited occurrence and R. In both cases, the *range* of v is r. Notice that the range of v is not in the scope of v.

We shall need to know that the semantics of terms and rules can be defined by first-order formulas using only primitive set-theoretic notions ($\in$ and Atoms), the adjacency relation of the input graph, and the dynamic functions. More precisely, consider any particular state of one of our ASM's. It is a structure H^+ with underlying set $HF(I)$ and with interpretations for all the function symbols listed in Definition 3. Let H be the structure

$$H = \langle HF(I), \in, I, A \rangle$$

that is like H except that among the non-logical symbols only $\in$, Atoms, and A are interpreted. (In the usual terminology of mathematical logic, H is a reduct of H^+.) Let H^D be the intermediate structure in which the dynamic function symbols are also interpreted (by the same functions as in H^+). We shall need to know that all essential aspects of the execution of Π in the state H^+, i.e., the computation leading from H^+ to its sequel, can be defined in the structure H^D. (It will turn out that, for every state H^+ that actually arises during the computation of a BGS machine, the dynamic functions will be definable in H, and so we could use H instead of H^D here. See the proof of Proposition 11 below.)

The longer version of this paper will contain these H^D-definitions in full; here we give only some typical examples. To avoid having to introduce new symbols for a multitude of set-theoretic formulas, we adopt the notational convention that $\lceil\varphi\rceil$ means the set-theoretic formalization of the (informal) statement φ.

We first define $\lceil y \in t\rceil$ and $\lceil y = t\rceil$ for all terms t; here y is a variable not free in t. Here are some typical clauses from the definition.

- If f is a dynamic function symbol, then $\lceil y = f(t_1, \ldots, t_j)\rceil$ is

$$\exists z_1 \ldots \exists z_j \left(\bigwedge_{i=1}^{j} \lceil z_i = t_i\rceil \wedge y = f(z_1, \ldots, z_j) \right).$$

- $\lceil y \in \mathrm{Pair}(t_1, t_2)\rceil$ is $\lceil y = t_1\rceil \vee \lceil y = t_2\rceil$.
- $\lceil y \in \{t(v) : v \in r : \varphi(v)\}\rceil$ is

$$\exists v \, (\lceil v \in r\rceil \wedge \lceil \mathtt{true} = \varphi(v)\rceil \wedge \lceil y = t(v)\rceil).$$

Next, we define in H^D the semantics of rules. For each rule R and each dynamic function symbol f, say of arity j, we first define a preliminary formula, $\lceil R$ wants to set f at $x_1, \ldots, x_j$ to $y\rceil$, which ignores possible conflicts between updates. This is a formula with free variables $x_1, \ldots, x_j, y$ and any variables $z_1, \ldots, z_k$ free in the rule R. It holds in state H^D of elements $a_1, \ldots, a_j, b, c_1 \ldots, c_k$ if and only if $((f, a_1, \ldots, a_j), b)$ is in the update set of rule $R(c_1, \ldots, c_k)$ in state H^+, as defined in [6]. (We use the symbol f in our name for the formula $\lceil R$ wants to set f at $x_1, \ldots, x_j$ to $y\rceil$, but f need not occur in the formula itself.) Typical clauses in the construction of $\lceil R$ wants to set f at $x_1, \ldots, x_j$ to $y\rceil$ include:

– $\lceil f(t_1, \ldots, t_j) := t_0$ wants to set f at $x_1, \ldots, x_j$ to $y\rceil$ is

$$\bigwedge_{i=1}^{j} \lceil x_i = t_i\rceil \wedge \lceil y = t_0\rceil.$$

– $\lceil$`do forall` $v \in r,\ R$ `enddo` wants to set f at $x_1, \ldots, x_j$ to $y\rceil$ is

$$\exists v\, (\lceil v \in r\rceil \wedge \lceil R \text{ wants to set } f \text{ at } x_1, \ldots, x_j \text{ to } y\rceil).$$

A rule may want to set f at $x_1, \ldots, x_j$ to several values. We adopt the standard ASM convention that if the program Π contains such a conflict, then all the dynamic functions remain unchanged. The formal definitions are as follows.

– $\lceil \Pi$ clashes$\rceil$ is

$$\bigvee_f \exists x_1 \ldots \exists x_j \exists y \exists z \\ (\Pi \text{ wants to set } f \text{ at } x_1, \ldots, x_j \text{ to } y) \wedge \\ (\Pi \text{ wants to set } f \text{ at } x_1, \ldots, x_j \text{ to } z) \wedge y \neq z.$$

Of course, the arity j depends on f.

– $\lceil \Pi$ sets f at $x_1, \ldots, x_j$ to $y\rceil$ is

$$\lceil \Pi \text{ wants to set } f \text{ at } x_1, \ldots, x_j \text{ to } y\rceil \wedge \neg\lceil \Pi \text{ clashes}\rceil.$$

Finally, we define the dynamic functions for the sequel state, that is, for the state obtained from H^+ by executing Π once.

For a j-ary dynamic function f, we define

$$\lceil y = f(x_1, \ldots, x_j) \text{ in the sequel}\rceil$$

to be

$$\lceil \Pi \text{ sets } f \text{ at } x_1, \ldots, x_j \text{ to } y\rceil \vee \\ (\lceil y = f(x_1, \ldots, x_j)\rceil \wedge \neg\exists y' \lceil \Pi \text{ sets } f \text{ at } x_1, \ldots, x_j \text{ to } y'\rceil$$

The preceding definitions provide most of the proof of the following result.

Proposition 11 *For any BGS program Π, there exists a number B with the following property. For each natural number m and each dynamic function symbol f, there is a first-order formula $\lceil y = f(x_1 \ldots, x_j)$ at step $m\rceil$ in the vocabulary $\{\in, Atoms, A\}$ such that, for any input structure (I, A), the tuples that satisfy $\lceil y = f(x_1 \ldots, x_j)$ at step $m\rceil$ in $H = \langle HF(I), \in, I, A\rangle$ constitute the graph of f in the m^{th} state of the run of Π on (I, A). Furthermore, the number of variables occurring in $\lceil y = f(x_1 \ldots, x_j)$ at step $m\rceil$ is at most B.*

It will be important that the bound B depends only on Π, not on m. To avoid possible confusion, we emphasize that variables can be re-used in these formulas; thus the same variable may be bound many times and may occur free as well.

Proof. We construct the required formulas by induction on m, starting with $\lceil y = f(x_1 \ldots, x_j)$ at step $0\rceil$, which can be taken to be $\lceil y = \emptyset\rceil$ because all dynamic functions are initialized to be constant with value $\emptyset$.

For the induction step from m to $m + 1$, we begin with the formula $\lceil y = f(x_1, \ldots, x_j)$ in the sequel$\rceil$ as constructed above. Then, for each dynamic function symbol g, we replace each occurrence of a subformula $\lceil t_0 = g(t_1, \ldots, t_k)\rceil$ with $\lceil t_0 = g(t_1, \ldots, t_k)$ at step $m\rceil$.

As for the bound B, it can be taken to be the maximum number of variables in any of the formulas $\lceil y = f(x_1, \ldots, x_j)$ in the sequel$\rceil$ as f ranges over all the dynamic function symbols. We omit the verification of this; it is a fairly standard application of the idea of re-using variables. □

Remark 12 For the purpose of proving Theorem 2, it is important that the number of variables in the formulas $\lceil y = f(x_1, \ldots, x_j)$ at step $m\rceil$ be bounded independently of m, but it is not crucial that the formulas be finite. Formulas of the infinitary language $L_{\infty,\omega}$ would serve as well. An alternate approach to obtaining such formulas is to express $\lceil y = f(x_1, \ldots, x_j)$ at step $z\rceil$ (with a variable z for the number of steps, numbers being viewed as von Neumann ordinals) in first-order logic with the least-fixed-point operator, and then to use the known translation from this logic into the infinitary, finite-variable logic $L^{\omega}_{\infty,\omega}$ (see [5]). This is the approach used in [2]. Then, to get the corresponding formulas for specific m in place of z, one only has to check that each natural number m can be defined with a fixed number of variables, independent of m. In fact, each natural number can be defined with just three variables.

We shall need a slight generalization of the notions, defined in [2], of "critical" and "active" elements of a state of a BGS computation. Instead of considering only states, we consider *pebbled states* consisting of a state together with an assignment of values to finitely many variables. (Pebbled states are the contexts in which it makes sense to evaluate a term or rule.) When the relevant variables and their ordering are understood, we think of a pebbled state as a state plus a tuple of elements, $(H, a_1, \ldots, a_j)$, where a_i is the value assigned to the i^{th} variable. For brevity, we sometimes use vector notation $\boldsymbol{a}$ for $(a_1, \ldots, a_j)$.

Definition 13 The *critical* elements of a pebbled state $(H, \boldsymbol{a})$ are

- all the atoms and the set I of atoms,
- the Boolean values `true` and `false`,
- all values of dynamic functions,
- all components of locations where dynamic functions have values other than $\emptyset$, and
- all components of $\boldsymbol{a}$.

An element is *active* if it is in the transitive closure of some critical element.

For ordinary (unpebbled) states, this definition differs from that in [2] only in that the set I of atoms is critical and therefore also active.

We are now in a position to give the example, promised in Section 2, of a BGS program Π together with polynomial bounds on the number of steps and the number of active elements, such that not all of the following three classes of graphs have asymptotic probability 0 or 1: the graphs on which Π halts with output `true` (within the prescribed bounds on steps and active elements), the analogous class for `false`, and the class of graphs on which Π fails to halt within the prescribed bounds. The required Π can be taken to be

```
do forall x ∈ Atoms
do forall y ∈ Atoms
   do in parallel
      if A(x, y) then f(Pair(x, y)) := true,
      Output := true,
      Halt := true
   enddo
enddo enddo
```

This program Π only executes once before halting, so we can take the polynomial bound on the number of steps to be 2 and ignore this bound. The number of active elements is $n+3+e$ where n and e are the numbers of vertices and edges in the input graph. (The active elements are the n atoms, the e two-element sets corresponding to edges, the two boolean values, and the set I of atoms.) In a large random graph, the expected value of e is $n(n-1)/4$, i.e., half the number of possible edges, but small fluctuations about this value are probable. Indeed, the asymptotic probability that $e \leq n(n-1)/4$ is $1/2$. So, if we impose a bound of $n + 3 + n(n-1)/4$ on the number of active elements, then with asymptotic probability $1/2$ our program will halt with output `true`, and with asymptotic probability $1/2$ it will fail to halt because it cannot execute its single computation step without activating too many elements.

4 Outline of Proof of Zero-One Law

We already know, from Proposition 11, that whether a BGS program halts at a particular step with a particular output can be defined by a first-order sentence

over the structure $H = \langle HF(I), \in, I, A\rangle$, with a number of variables that does not depend on the number of steps. A natural approach to proving Theorem 2 would therefore be to use Ehrenfeucht-Fraïssé games and produce winning strategies for the duplicator, to show that such sentences have the same truth value for almost all input graphs (I, A). Unfortunately, that isn't true; for example, the parity of $|I|$ can be defined by a first-order statement over H.

As in [2], this approach can be modified by defining a subclass S of $HF(I)$ for which the duplicator has the necessary winning strategies, and then showing that the definitions given in Proposition 11 remain correct when interpreted in S instead of $HF(I)$. S consists of those elements of $HF(I)$ that have suitable symmetry properties. In [2], symmetry meant invariance under enough automorphisms of the input structure. But almost all graphs have no non-trivial automorphisms, so a subtler approach to symmetry is needed. Shelah introduces a suitable class of partial automorphisms (for any given program Π and any given polynomial bound on the number of active elements) and shows that it leads to an appropriate notion of symmetry. Here "appropriate" means that the symmetry requirements are restrictive enough to provide winning strategies for the duplicator yet are lenient enough to include all the sets actually involved in a computation of Π, limited by the given bound on active elements.

The hardest part of the proof is the leniency just mentioned: The computation involves only symmetric sets. This will be proved by a double induction, first on the stages of the computation and second, within each stage, on the subterms and subrules of Π. That inner induction proceeds along a rather unusual ordering of the subterms and subrules, which we call the computational ordering.

In this double induction, it is necessary to strengthen the induction hypothesis, to say not only that every set x involved in the computation is symmetric but also that all sets x' obtained from x by applying suitable partial automorphisms are also involved in the computation. The assumed bound on the number of active elements will imply a polynomial bound on the number of involved elements. (Not all involved elements are active, but there is a close conection between the two.) That means that the number of x''s is limited, which in turn implies, via a highly non-trivial combinatorial lemma, that x is symmetric.

The traditional extension axioms, as in [5], are satisfied by almost all graphs and are adequate to produce the duplicator's strategies that we need, but they are not adequate to imply the combinatorial lemma needed in the symmetry proof. For this purpose, we need what we call strong extension axioms, saying that every possible type over a finite set is not only realized but realized by a large number of points.

In the next few sections, we shall assemble the tools for the proof that have been mentioned here. After thus describing the proof in somewhat more detail, we shall add some sections about related issues.

5 Tasks and Their Computational Order

To compute the value of a term or the update set of a rule, one needs a structure (the state of the computation) and values for the variables free in the term or rule. In most of our discussion, there will be a fixed structure under consideration but many choices of values for variables. It will therefore be convenient to push the structures into the background, often neglecting even to mention them, and to work with pairs whose first component is a term or rule and whose second component is an assignment of values to some variables (including all the variables free in the first component). We shall call such pairs "tasks" because we regard them as things to be evaluated in the course of a computation.

The official definition of tasks will differ in two respects from this informal description. First, we shall always have a particular program Π under consideration, and we deal only with terms and rules occurring in Π. Recall in this connection our Convention 9, by which we are really dealing with occurrences of terms and rules. Second, it will be convenient to include in our tasks not only the obviously necessary values for free variables but also values for certain additional variables, defined as follows.

Definition 14 A variable v is *pseudo-free* in a term t or rule R if t or R lies in the scope of v.

Because Π, being a program, has no free variables, all the free variables in a term or rule must also be pseudo-free in it. But there may be additional pseudo-free variables. From a syntactic point of view, v is pseudo-free in t or R if one could introduce v as a free variable in t or R without violating the requirement that Π have no free variables. From a computational point of view, the pseudo-free variables of t or R are those variables to which specific values have been assigned whenever one is about to evaluate t or R in the natural algorithm for executing Π.

Definition 15 A *term-task* (relative to a program Π and a state H) is a pair $(t, \boldsymbol{a})$ where t is a term in Π and $\boldsymbol{a}$ is an assignment of values in H to all the pseudo-free variables of t. *Rule-tasks* are defined similarly except that the first component is a rule in Π. Term-tasks and rule-tasks are collectively called *tasks*.

Although the second component $\boldsymbol{a}$ of a task is officially a function assigning values to the pseudo-free variables of the first component, we sometimes view it as simply a tuple of values. This makes good sense provided we imagine a fixed order for the pseudo-free variables. We also sometimes write $\boldsymbol{a}\restriction$ for the restriction of the function $\boldsymbol{a}$ to an appropriate subset of its domain; it will always be clear from the context what the appropriate subset is.

We write $\mathrm{Val}(t, \boldsymbol{a})$ for the value of the term t in a structure (assumed fixed throughout the discussion) when the free variables are assigned values according to $\boldsymbol{a}$.

Definition 16 The *computational order* $\prec$ is the smallest transitive relation on tasks satisfying the following conditions.

- $(t_i, \boldsymbol{a}) \prec (f(t_1, \ldots, t_j), \boldsymbol{a})$ for $1 \leq i \leq j$.
- $(r, \boldsymbol{a}) \prec (\{s : v \in r : \varphi\}, \boldsymbol{a})$ and, for each $b \in \mathrm{Val}(r, \boldsymbol{a})$, all three of $(v, \boldsymbol{a}, b)$, $(s, \boldsymbol{a}, b)$ and $(\varphi, \boldsymbol{a}, b)$ are $\prec (\{s : v \in r : \varphi\}, \boldsymbol{a})$.
- $(t_i, \boldsymbol{a}) \prec (f(t_1, \ldots, t_j) := t_0)$ for $0 \leq i \leq j$.
- All of $(\varphi, \boldsymbol{a})$, $(R_0, \boldsymbol{a})$, and $(R_1, \boldsymbol{a})$ are $\prec$ (`if` φ `then` R_0 `else` R_1 `endif`, $\boldsymbol{a}$).
- $(r, \boldsymbol{a}) \prec$ (`do forall` $v \in r,\ R$ `enddo`, $\boldsymbol{a}$) and, for each $b \in \mathrm{Val}(r, \boldsymbol{a})$, both $(v, \boldsymbol{a}, b)$ and $(R, \boldsymbol{a}, b)$ are $\prec$ (`do forall` $v \in r,\ R$ `enddo`, $\boldsymbol{a}$).
- If r is the range of a variable v pseudo-free in t or R, then $(r, \boldsymbol{a}\restriction) \prec (t, \boldsymbol{a})$ or $(r, \boldsymbol{a}\restriction) \prec (R, \boldsymbol{a})$, respectively.

Except for the last clause, the definition assigns as the predecessors of a task simply the subterms or subrules of its first component, equipped with suitable values for the variables. The last clause, however, is quite different. Here the first component r of the lower clause may well be much larger than the first component t or R of the upper clause.

Intuitively, if one task precedes another in this computational ordering, then in the natural calculation involved in the execution of Π the former task would be carried out before the latter. In the range clause, the idea is that, before attempting to evaluate t or R, we would have assigned a value to v, and before that we would have evaluated the range in order to know what the possible values for v are.

This intuition strongly suggests that the computational order should be well-founded. In fact it is, but the proof is not trivial and will only be sketched here. To treat term-tasks and rule-tasks simultaneously, we use $X, Y, \ldots$ to stand for either terms t or rules R.

Proposition 17 *There is a rank function ρ, mapping terms and rules to natural numbers, such that if $(X, \boldsymbol{a}) \prec (Y, \boldsymbol{b})$ then $\rho(X) < \rho(Y)$.*

Proof. Let V be the number of variables occurring in Π, and let D be the maximum depth of nesting of terms and rules occurring in Π.

For each variable v in Π, let $\tau(v)$ be the number of symbols in the term or rule that binds v. Notice that, if x is bound in the range of y then $\tau(x) < \tau(y)$.

Call a variable *relevant* to a term or rule X if it is either pseudo-free in X or bound in X. In other words, the scope of v either includes or is included in X. Define $\sigma(X)$ to be the sum of $V^{\tau(v)}$ over all variables relevant to X. As one goes downward in the computational order, using any clause in its definition except the range clause, the set of relevant variables either remains the same or shrinks, so the value of σ remains the same or decreases. Furthermore, when σ stays the same, the depth of nesting inside the term or rule decreases. As one goes downward using the range clause, the variable v explicitly mentioned in the clause loses its relevance, but other variables, bound in r, may become relevant. All of the latter have τ values strictly smaller than that of v, and there are fewer than V of them, so σ still decreases.

Therefore, if we define

$$\rho(X) = (D + 1)\sigma(X) + \mathrm{depth}(X)$$

where depth(X) means the depth of nesting of terms and rules in X, then $\rho(X)$ decreases at every downward step in the computational ordering. □

This proposition immediately implies that the computational order is a strict partial order; it has no cycles. Since finite partial orders are well-founded, it is legitimate to do proofs by induction with respect to $\prec$. Such proofs can also be regarded as ordinary mathematical induction on the rank ρ. Furthermore, induction on ρ is somewhat more powerful than induction with respect to $\prec$ because ρ-induction allows us, when proving a statement for some task, to assume the same statement not only for all computationally earlier tasks $(X, \boldsymbol{a})$ but also for all tasks $(X, \boldsymbol{b})$ having the same first components as these. This is because ρ depends only on the first component.

The following lemma is useful technically, and it also serves to clarify the role of the range clause in the definition of the computational ordering. The definition of this ordering $\prec$ ensures that, if one task T precedes another T' then there is a chain

$$T' = T_0 \succ T_1 \succ \cdots \succ T_n = T$$

joining them, in which each two consecutive terms are related as in one of the clauses in Definition 16.

Lemma 18 *If $T' \succ T$ then there is a chain as above, in which the range clause is used, if at all, only at the first step, from T_0 to T_1.*

We omit the proof, which is an induction on the length n of the chain.

Corollary 19 *If a task T precedes a task T' with no pseudo-free variables in its first component (and thus with empty second component), then this can be established without the range clause.*

Proof. The chain from T' down to T obtained in the lemma must not use the range clause at all, for the range clause is not applicable at the first step in the absence of pseudo-free variables in T'. □

The important case of the corollary is when (the first component of) T' is the whole program Π.

6 Involved and Active Elements

Throughout this section, we assume that we are dealing with a fixed program Π and a fixed state arising during its execution on some input (I, A). As before, we write H for the structure $\langle HF(I), \in, I, A\rangle$. The state under consideration is an expansion of H, interpreting the dynamic function symbols as well as the other static function symbols. We write H^+ for this state. Notice that, by Proposition 11, the interpretations of the additional function symbols of H^+ are all definable in H. The definitions of the dynamic function symbols depend on the stage of the computation; the others do not.

The following definition is intended to capture the intuitive notion of an element of $HF(I)$ being "looked at" during the execution of a task T. It doesn't quite fulfill this intention, for it includes too many elements, pretending for example that execution of an `if ... then ... else ...` includes execution of both the consitituent rules regardless of the truth value of the guard. Nevertheless, it serves our purposes because (1) it includes all the elements that are really needed (see Lemma 22 below) and (2) it doesn't include too many additional elements (see Lemma 23 below).

Definition 20 An element $c \in HF(I)$ is *involved* in a task $(X, \boldsymbol{a})$ (with respect to a program Π and a state H^+) if either it is active in $(H^+, \boldsymbol{a})$ or it is the value in H^+ of some term-task $\prec (X, \boldsymbol{a})$.

The next three lemmas give the key properties of this notion of "involved."

Lemma 21 *Any object that is not active in a state H^+ but is active in its sequel with respect to Π must be involved in the task Π with respect to state H^+.*

Lemma 22 *The set-theoretic definitions at the end of Section 3, in particular Proposition 11, remain correct if the quantifiers are interpreted to range only over elements involved in Π rather than over all of $HF(I)$.*

This is proved by inspecting all those definitions, including the ones that we did not explicitly exhibit, seeing which values of the quantified variables are essential, and checking that these values are all involved in Π.

Lemma 23 *For any state H^+ and any task $(X, \boldsymbol{a})$, the number of elements involved in $(X, \boldsymbol{a})$ is bounded by a polynomial function of the number of active elements in $(H^+, \boldsymbol{a})$. The polynomial depends only on the term or rule X.*

This is proved by induction on X. We are interested in the lemma primarily in the case that the task $(X, \boldsymbol{a})$ is the entire program Π (with the empty assignment, as Π has no pseudo-free variables). A $\tilde{\text{C}}$PTime program comes with a polynomial bound on the number of active elements during its run; the lemma allows us to deduce a (possibly larger) polynomial bound on the number of involved elements. This will be crucial in showing that the computation takes place entirely in the world of symmetric sets.

7 Strong Extension Axioms

Let $\tau(x_0, x_1, \ldots, x_k)$ be a quantifier-free formula of the form

$$\Big(\bigwedge_{1 \le i < j \le k} x_i \neq x_j \Big) \to \Big(\bigwedge_{1 \le i \le k} (x_0 \neq x_i) \wedge \pm(x_0 A x_i) \Big),$$

so that, for a given k, there are exactly 2^k different formulas of that form. The extension axiom $\mathrm{EA}(\tau)$ is the axiom $\forall x_1 \ldots x_k \exists x_0 \tau(x_0, \ldots, x_k)$. A graph G satisfies the *strong extension axiom* $SEA(\tau)$, if for all distinct vertices $x_1, \ldots, x_k$,

there are at least $\frac{1}{2}n/2^k$ vertices x_0 in G satisfying $\tau(x_0, x_1, \ldots, x_k)$. The extension axiom EA_k is the conjunction of all 2^k extension axioms $\mathrm{EA}(\tau)$ with $k+1$ variables; the strong extension axiom SEA_k is the conjunction of all 2^k strong extension axioms $\mathrm{SEA}(\tau)$ for all τ with $k+1$ variables.

Thus, EA_k says that every possible configuration for a vertex x_0, relative to k given vertices $x_1, \ldots, x_k$, actually occurs. SEA_k says that every such configuration occurs fairly often.

Why $\frac{1}{2}n/2^k$? In a random graph with n vertices, the probability that an arbitrary vertex a_0, different from $a_1, \ldots, a_k$, satisfies $\tau(a_0, a_1, \ldots, a_k)$ is $1/2^k$, so the expected number of vertices a_0 satisfying $\tau(a_0, a_1, \ldots, a_k)$ is $(n-k)/2^k$. So with high probability, there are at least $\frac{1}{2}n/2^k$ vertices a_0 in G satisfying $\tau(a_0, a_1, \ldots, a_k)$. The factor $\frac{1}{2}$ could be replaced with any positive constant $c < 1$; that gives a strong extension axiom SEA^c_k.

Lemma 24 *For each k, the asymptotic probability of SEA_k is 1.*

The proof uses Chernoff's inequality from probability theory.

8 Supports

In this section, we describe the notion of symmetry with respect to partial automorphisms that will, as explained in Section 4, apply to all objects involved in a computation and lead to winning strategies for the duplicator in certain Ehrenfeucht-Fraïssé games. Two numerical parameters will be involved in this notion of symmetry, namely the size of the partial automorphisms and the number of atoms a symmetric object can depend on. The appropriate values for these parameters will depend on the BGS program and the polynomial bound on the number of active elements.

For the purposes of this section, let $q \geq 1$ and $k \geq 4$ be fixed integers. When this material is applied in the proof of Theorem 2, q will be the degree of a polynomial bounding the number of involved elements (obtainable from Π and the bound on active elements, via Lemma 23) and k will be $2B+4$, where B is as in Proposition 11.

We assume that the input graph (I, A) satisfies the extension axioms for up to $3kq$ variables. (We don't need the strong extension axioms yet.)

For brevity we adopt the conventions that

- w, x, y, z (possibly with subscripts, superscripts, or accents) stand for members of $HF(I)$,
- a, b, c (possibly with subscripts, superscripts, or accents) stand for sets of $\leq q$ atoms, and we call such sets *possible supports*, and
- $\alpha, \beta, \gamma, \delta, \zeta$ (possibly with subscripts or superscripts) stand for partial automorphisms of the graph (I, A) whose domains have

The inverse α^{-1} of a motion and the composite $\alpha \circ \beta$ of two motions are defined in the obvious way. In particular, the domain of $\alpha \circ \beta$ is $\beta^{-1}(\mathrm{Dom}(\alpha))$.

The extension axioms imply that, if α is a motion and s is a set of atoms with $|\mathrm{Dom}(\alpha)| + |s| \leq kq$, then α can be extended to a motion whose domain includes s. (In fact, the extension axioms imply considerably more, as they go up to $3kq$ variables, not just the kq needed for the preceding statement.)

We next define, by simultaneous recursion on the rank of x, the three concepts "a supports x," "x is supported," and "$\hat{\alpha}(x)$," where the last of these is defined if and only if $\mathrm{Dom}(\alpha)$ includes some a that supports x.

Definition 25 If x is an atom, then

- a supports x if and only if $x \in a$,
- x is supported (always), and
- $\hat{\alpha}(x) = \alpha(x)$.

If, on the other hand, x is a set, then

- a supports x if and only if every $y \in x$ is supported and, for every y of lower rank than x and every motion α, if α pointwise fixes a and if $\mathrm{Dom}(\alpha)$ includes some set supporting y, then

$$y \in x \iff \hat{\alpha}(y) \in x,$$

- x is supported if and only if some a supports x, and
- if a supports x and $a \subseteq \mathrm{Dom}(\alpha)$, then $\hat{\alpha}(x)$ is the set of all $\hat{\beta}(y)$ where $y \in x$, $\beta \restriction a = \alpha \restriction a$, and $\mathrm{Dom}(\beta)$ includes some support of y.

The definition of $\hat{\alpha}(x)$ when x is a set seems to depend on the choice of a particular support a of x. The first part of the following lemma gets rid of that apparent dependence; the rest of the lemma gives useful technical information about supports and about the application of motions to supported sets.

Lemma 26 *1. $\hat{\alpha}$ is well-defined. Specifically, if a_1 and a_2 both support x and are both included in $Dom(\alpha)$, then $\hat{\alpha}^1(x)$ and $\hat{\alpha}^2(x)$, defined as above using a_1 and a_2 respectively, are equal.*

2. *If a supports x and $a \subseteq Dom(\alpha) \cap Dom(\beta)$ and $\alpha \restriction a = \beta \restriction a$, then $\hat{\alpha}(x) = \hat{\beta}(x)$.*
3. *If $\hat{\alpha}(x)$ is defined then it has the same rank as x.*
4. *If α is an identity map, then so is $\hat{\alpha}$, i.e., $\hat{\alpha}(x) = x$ whenever $\hat{\alpha}(x)$ is defined.*
5. *$\hat{\alpha}$ is a partial automorphism of the structure $(HF(I), \in, I, A)$. In other words, $\hat{\alpha}(I) = I$ and, whenever x and x' have supports included in $Dom(\alpha)$,*

$$\begin{aligned} x' = x &\iff \hat{\alpha}(x') = \hat{\alpha}(x) \\ x' \in x &\iff \hat{\alpha}(x') \in \hat{\alpha}(x) \\ x' A x &\iff \hat{\alpha}(x') A \hat{\alpha}(x). \end{aligned}$$

6. *If a supports x and $a \subseteq Dom(\alpha)$ then $\alpha[a]$ supports $\hat{\alpha}(x)$.*
7. *If $\hat{\alpha}(x)$ is defined and $a \subseteq Dom(\alpha)$ and $\alpha[a]$ supports $\hat{\alpha}(x)$, then a supports x.*

8. $\widehat{\beta \circ \alpha} = \hat{\beta} \circ \hat{\alpha}$ *in the following sense: If* $\widehat{\beta \circ \alpha}(x)$ *is defined then so is* $\hat{\beta}(\hat{\alpha}(x))$ *and they are equal.*

All eight parts of the lemma are proved together, by induction on the maximum of the ranks of x and x'. We omit the tedious proof.

Definition 27 S is the collection of supported objects in $HF(I)$. We also write S for the structure $(S, I, \in, A)$.

By the definition of supports, S is a transitive set containing all the atoms; by (5) of Lemma 26, it also contains I. Furthermore, by (5) and (6) of that lemma, each $\hat{\alpha}$ is a partial automorphism of S. In fact, as the following lemma shows, the $\hat{\alpha}$'s are much better than just partial automorphisms, because they fit together well. Recall that $L^k_{\infty,\omega}$ is the part of the infinitary first-order language $L_{\infty,\omega}$ (allowing infinite conjunctions and disjunctions) consisting of formulas with at most k variables (free or bound, but the same variable can be re-used). Recall also that $k \geq 4$ is fixed throughout this section.

Lemma 28 *Let* φ *be a formula of* $L^k_{\infty,\omega}$ *with* $j \leq k$ *free variables. Let* α *be a motion, and let* $x_1, \ldots, x_j$ *be elements of* S *with supports included in* $Dom(\alpha)$. *Then*

$$S \models \varphi(x_1, \ldots, x_j) \iff S \models \varphi(\hat{\alpha}(x_1), \ldots, \hat{\alpha}(x_j)).$$

Proof. We give a strategy for the duplicator in the Ehrenfeucht-Fraïssé game for $L^k_{\infty,\omega}$. At any stage of the game, let $\boldsymbol{y}$ and $\boldsymbol{z}$ be the positions of the pebbles on the two boards; so initially, $y_i = x_i$ and $z_i = \hat{\alpha}(x_i)$. The duplicator's strategy is to arrange that there is always a motion β whose $\hat{\beta}$ sends each y_i to the corresponding z_i. There is such a β initially, namely α, and as long as he maintains such a β the duplicator cannot lose, by (5) of Lemma 26. So we need only check that, if such a β exists and then the spoiler moves, the duplicator can move so that again a (possibly new) β does the job. Without loss of generality, suppose the spoiler moves the first pebble on the left board from its position y_1 to a new $y'_1 \in S$. Restrict the old β to the union of supports of the y_i for $i \neq 1$. There are strictly fewer than k of these supports, hence at most $(k-1)q$ points in the domain of the restricted β. So we can extend this motion to a new β having in its domain some support of the new y'_1. The resulting $\hat{\beta}(y'_1)$ is where the duplicator should move the pebble from z_1. The resulting board position and the new β satisfy the specification of the duplicator's strategy (thanks to (2) of Lemma 26). So we have shown that the duplicator can carry out the indicated strategy. □

We shall need variants of these results, dealing with two graphs and the universes of hereditarily finite sets built over them. Specifically, suppose (I_1, A) and (I_2, A) are graphs satisfying the extension axioms for up to $3kq$ variables. (We've simplified notation slightly by using the same name A for the adjacency relations in both graphs I_i.) For $i, j \in \{1, 2\}$, we define an i, j-*motion* to be a partial isomorphism of size at most kq from I_i to I_j. Thus 1,1-motions and 2,2-motions are motions in the earlier sense for I_1 and I_2, respectively.

If α is a 1,2-motion, then we define $\hat{\alpha}(x)$ for all $x \in HF(I_1)$ having supports included in $\mathrm{Dom}(\alpha)$. We do this in exact analogy with the earlier definition: If x is an atom then $\hat{\alpha}(x) = \alpha(x)$. If x is a set with a support $a \subseteq \mathrm{Dom}(\alpha)$, then $\hat{\alpha}(x)$ is the set of all $\hat{\beta}(y)$ where $y \in x$, β is a 1,2-motion extending $\alpha \restriction a$, and $\mathrm{Dom}(\beta)$ includes some support of y.

Similarly, we define $\hat{\alpha}(x)$ when α is a 2,1-motion and $x \in HF(I_2)$ has a support $\subseteq \mathrm{Dom}(\alpha)$. These definitions together with the definitions already available from Section 8 for 1,1- and 2,2-motions allow us to refer to $\hat{\alpha}(x) \in HF(I_j)$ whenever α is an i,j-motion and $x \in HF(I_i)$ has a support included in $\mathrm{Dom}(\alpha)$.

We can now repeat the earlier arguments in this slightly more general context. No conceptual changes or additions are needed, only a little bookkeeping to keep track of the four different sorts of motions. We exhibit for future reference the 1,2-analog of Lemma 28. Let S_i be the collection of supported objects in $HF(I_i)$; as before, we also write S_i for the structure $(S_i, \in, I_i, A)$.

Lemma 29 *Let φ be a formula of $L^k_{\infty,\omega}$ with $j \leq k$ free variables. Let α be a 1,2-motion, and let $x_1, \ldots, x_j$ be elements of S_1 with supports included in $\mathrm{Dom}(\alpha)$. Then*

$$S_1 \models \varphi(x_1, \ldots, x_j) \iff S_2 \models \varphi(\hat{\alpha}(x_1), \ldots, \hat{\alpha}(x_j)).$$

9 Combinatorics

In this section, we describe (without proof) the main combinatorial lemma needed in the proof of the zero-one law. The parameters q and k are fixed as before, and the graph (I, A) of atoms is now assumed to satisfy the *strong* extension axioms for up to $3kq$ variables.

By a *polymer* we mean a sequence of at most kq atoms. (The reason for the terminology is that, in [2], we used "molecule" for a one-to-one listing of a support; here that would be a sequence of length q. A polymer is essentially the concatenation of up to k molecules. It gives the supports for up to k objects.) The *configuration* of a polymer consists of the following information: which components are equal and which are adjacent in the graph of atoms. If the polymer has length l, then its configuration could be viewed as the combination of an equivalence relation on $\{1, 2, \ldots, l\}$ (telling which components are equal) and an irreflexive symmetric relation on the quotient set (telling which components are adjacent). By the *joint configuration* of two (or more) polymers, we mean the equality and adjacency information about all components of both (or all) of the polymers. It could be viewed as the configuration of the concatenation of the polymers, except for the technicality that the concatenation may be too long to count as a polymer.

We shall be concerned with equivalence relations of the following sort.

Definition 30 A *configuration-determined equivalence* (*cde* for short) is an equivalence relation E with the following two properties.

– Its domain consists of all polymers of one specified configuration.

– Whether two polymers ξ and η of this configuration are related by E depends only on their joint configuration.

When dealing with polymers ξ of a specified configuration (e.g., those in the domain of a cde), we can simplify their presentation by omitting any repetitions of components in ξ. Because of the tight correspondence between the polymers ξ of a known configuration and these compressed versions, we can confine our attention to the compressed versions; that is, we can assume that we deal only with polymers that are one-to-one sequences.

Theorem 31 *Assume that* $|Atoms| \geq kq2^{3kq+1}$. *Let* E *be a configuration-determined equivalence with fewer than*

$$\frac{1}{(q+1)!}\left(\frac{|Atoms|}{2^{3kq+1}}\right)^{q+1}$$

equivalence classes. Let l *be the common length of the polymers in the domain of* E. *There exists a set* $u \subseteq \{1, 2, \ldots, l\}$ *of size at most* q, *and there exists a group* G *of permutations of* u *such that, for any* ξ *and* η *in the domain of* E,

$$\xi E \eta \iff (\exists \sigma \in G)\,(\forall i \in u)\,\xi_i = \eta_{\sigma(i)}.$$

Notice that, although l can be as large as kq, the theorem requires u to be relatively small, of size at most q.

The conclusion of the theorem completely describes E in terms of u and G. It says that the E-equivalence class of ξ consists of those polymers (of the right configuration) obtainable from ξ by

– permuting the components indexed by u, using a permutation from G, and
– changing the other components completely arbitrarily.

In particular, the equivalence class of ξ depends only on the ξ_i for $i \in u$.

The hypothesis of the theorem involves a complicated bound on the number of equivalence classes. Most of the complication disappears if one remembers that q and k are fixed, so the bound is, up to a constant factor, just $|\text{Atoms}|^{q+1}$. When we apply the theorem, the cde's of interest will be such that the equivalence classes correspond to elements involved in the computation, so their number is bounded by a polynomial in |Atoms| of degree at most q. So the bound will automatically be satisfied once the number of atoms is large enough.

Because of space and time limitations, we omit Shelah's proof of this combinatorial theorem.

10 Putting the Proof Together

Consider a BGS program Π and a polynomial bound on the number of active elements. According to Lemma 23, we obtain a polynomial bound on the number of elements involved in Π at any stage of the computation. Let q be the degree

of this polynomial. Also, let $k = 2B + 4$ where B is the bound from Proposition 11 on the number of variables in the set-theoretic formulas describing the computation of Π. Whenever we refer to supports, motions, etc., we take these concepts to refer to the particular q and k just introduced.

In the following theorem, to say that a formula is absolute for S means that the formula's meaning does not change if the quantifiers are interpreted as ranging over the class S of supported objects rather than over all of $HF(I)$.

Theorem 32 *Assume that all active elements in state H^+ are supported. Then the following are true for every term-task $(t, \boldsymbol{a}) \prec \Pi$, provided the input graph (I, A) satisfies the strong extension axioms up to $3kq$ variables and is large enough.*

1. *If $(t, \boldsymbol{a}) \prec (X, \boldsymbol{a}')$ and if α is a motion whose domain includes supports of all the elements of $\boldsymbol{a}$, then $(t, \hat{\alpha}(\boldsymbol{a})) \prec (X, \hat{\alpha}(\boldsymbol{a}'))$.*
2. *All elements of $\boldsymbol{a}$ and of $Val(t, \boldsymbol{a})$ are supported.*
3. *The formula defining $x \in Val(t, \boldsymbol{y})$ is absolute for S when $\boldsymbol{y}$ is instantiated to $\boldsymbol{a}$.*
4. *$Val(t, \boldsymbol{a})$ is supported. Furthermore, given supports for all components of $\boldsymbol{a}$, their union includes a support for $Val(t, \boldsymbol{a})$.*
5. *The formula defining $x = Val(t, \boldsymbol{y})$ is absolute for S when $\boldsymbol{y}$ is instantiated to $\boldsymbol{a}$.*

All five parts of the theorem are proved in a simultaneous induction on $(t, \boldsymbol{a})$ with respect to the rank function ρ from Proposition 17 the computational order. The proof is too long to give here, but we describe a few points in it that explain why much of the work in previous sections is needed.

The computational order, and specifically the range clause in its definition, are crucial in the proof of (1). Consider, for example, a situation where $\boldsymbol{a} = (\boldsymbol{c}, b)$ and $(t, \boldsymbol{c}, b) \prec (Y, \boldsymbol{c})$ where b is the value assigned to a variable v bound by Y and thus pseudo-free in the subterm t. So $b \in \mathrm{Val}(r, \boldsymbol{c})$, where r is the range of v. To show that $(t, \hat{\alpha}(\boldsymbol{c}), \hat{\alpha}(b)) \prec (Y, \hat{\alpha}(\boldsymbol{c}))$, we want to know that $\hat{\alpha}(b) \in \mathrm{Val}(r, \hat{\alpha}(\boldsymbol{c}))$. Fortunately, the range clause makes $(r, \boldsymbol{c})$ a computational predecessor of $(t, \boldsymbol{c}, b)$, so we can apply induction hypotheses (2) and (3) to it. Thus, the fact that $b \in \mathrm{Val}(r, \boldsymbol{c})$ can be expressed as a set-theoretic statement true in S. And this statement will retain its truth value when we apply $\hat{\alpha}$, by Lemma 28.

The combinatorial Theorem 31 is used in the hardest case in the proof of (4), namely where t is $\{s : w \in r : \varphi\}$. Choose supports for all the a_i and let ξ_0 be a polymer in which all those supports are listed. For any other polymer ξ of the same configuration as ξ_0, let α be the motion sending ξ_0 to ξ. Write $t(\xi)$ for $\mathrm{Val}(t, \hat{\alpha}(\boldsymbol{a}))$. In particular, $t(\xi_0)$ is the object $\mathrm{Val}(t, \boldsymbol{a})$ that we hope to prove to be supported.

Define an equivalence relation E on the set of polymers of the same configuration as ξ_0 by

$$\xi E \xi' \iff t(\xi) = t(\xi').$$

One can verify, using the induction hypotheses and Lemma 28, that this E is configuration-determined. The number of equivalence classes of E is the number

of different elements of the form $t(\xi) = \mathrm{Val}(t, \hat{\alpha}(\boldsymbol{a}))$. It follows from part (1) that the number of such elements is bounded by a constant times $|I|^q$. That's smaller than the bound required in the Dichotomy Theorem, a polynomial of degree $q+1$, provided $|I|$ is large enough. So applying the combinatorial Theorem 31, we find that $t(\xi)$ depends only on the restriction of ξ to a certain set u of size at most q. Then one can show that the range of $\xi_0 \restriction u$ supports $\mathrm{Val}(t, \boldsymbol{a})$.

Corollary 33 *Assume that all active elements in state H^+ are supported. Then so are all objects involved in Π with respect to H^+.*

This is immediate from part (4) of the theorem and the definition of "involved." By Lemma 21, it implies that every active element of the sequel is supported, and so the theorem and its corollary are applicable to the sequel. Proceeding inductively and then invoking Lemma 22, we find that the set-theoretic formulas describing the computation are all absolute for S as long as the polynomial bound on the number of active elements is obeyed. But then, thanks to Lemma 29, the truth values of these formulas and therefore the behavior of the computation are the same for all input graphs that satisfy the necessary strong extension axioms and are sufficiently large. By virtue of Lemma 24, the class of such graphs has asymptotic probability 1, so Theorem 2 is proved.

11 Extension and Strong Extension

The zero-one law for first-order logic is based on the extension axioms: for every first-order sentence φ (of relational vocabulary), there exists k such that EA_k implies φ or EA_k implies $\neg\varphi$. The same holds for fixed-point logic FO+LFP and for the infinitary logic $L^{\omega}_{\infty,\omega}$ [5]. However, extension axioms are too weak to support the zero-one law for $\tilde{\mathrm{C}}$PTime. We give an example of a single polynomial time BGS program that separates structures satisfying arbitrarily many extension axioms. So strong extension axioms are really needed for the $\tilde{\mathrm{C}}$PTime zero-one law.

Example 34 For simplicity, we consider graphs equipped with a unary relation, (I, A, R). We informally describe a BGS program computing the maximal size of a clique included in R.

- In mode Initial, initialize i, p to 0, initialize C to $\{\emptyset\}$, and go to mode Compute. Intuitively, i is a counter, p is the parity of i and C is the collection of all subsets of R of size i.
- In mode Compute, increase i by one, flip p, update C as follows

  $$C := \{x \cup \{y\} : x \in C \wedge y \in R\}$$

  and go to mode Decide.
- In mode Decide, check if C contains a clique. If yes then go to mode Compute; otherwise output $1 - p$ and halt.

Consider running this program on the input having vertex set $\{1, 2, \ldots, n\}$, a random adjacency relation A, and R interpreted as $\{1, 2, \ldots, r\}$ for some $r < n$. It follows from results in [4, Section XI.1] that there are values of r of magnitude roughly $\log n \cdot \log \log n$ for which the clique number of $(R, A \restriction R)$ is very probably even and there are other nearby values of r for which this clique number is very probably odd. Also, because r is so much smaller than n (and the cliques smaller yet) the computation of Π will very probably activate fewer than n sets. So we can impose a polynomial bound slightly larger than $2n$ and be reasonably certain that the computation will halt. Finally, by choosing r and therefore n large enough, one can ensure (again with very high probability) that the graphs under consideration satisfy any specified extension axiom EA_k.

Since the strong extension axioms go beyond the ordinary extension axioms, one might hope that they imply some of the classical properties of almost all graphs, like rigidity and hamiltonicity, that are known [3] not to follow from extension axioms.

This is not the case for rigidity. The non-rigid graphs constructed in [3] — random modulo an imposed symmetry — can be shown to also satisfy strong extension axioms.

For Hamiltonicity, the situation is less clear. We can show, again using the construction from [3], that the axioms SEA_k^c for $c < \frac{1}{2}$ do not imply Hamiltonicity. For $c \geq \frac{1}{2}$, these examples no longer work, but we do not know whether others are available.

12 The Almost Sure Theory Is Undecidable

In the case of first-order logic, the almost sure theory (that is the set of almost surely true sentences) is decidable. The same holds if we add the least fixed point operator to first-order logic [1]. But it fails for $\tilde{\mathrm{C}}$PTime.

Proposition 35 *The class of almost surely accepting polynomially bounded programs and the class of almost surely rejecting polynomially bounded programs are recursively inseparable.*

Proof. Consider Turing machines with two halting states h_1 and h_2. For $i = 1, 2$, let H_i be the collection of Turing machines that halt in state h_i on the empty input tape. It is well-known that H_1 and H_2 are recursively inseparable. Associate to each Turing machine T a polynomial time BGS program as follows. The program Π ignores its input graph and simulates T on empty input tape (working exclusively with pure sets). Π outputs `true` (resp. `false`) if T halts in state h_1 (resp. h_2). The polynomial bounds on steps and activated elements are both the identity function, i.e., the number of atoms. Then if $T \in H_1$ (resp. $T \in H_2$) our polynomial time BGS program will accept (resp. reject) all sufficiently large inputs. □

References

1. Andreas Blass, Yuri Gurevich, and Dexter Kozen, *A zero-one law for logic with a fixed-point operator*, Information and Control 67 (1985) 70–90.
2. Andreas Blass, Yuri Gurevich, and Saharon Shelah, *Choiceless polynomial time*, Ann. Pure Applied Logic 100 (1999) 141–187.
3. Andreas Blass and Frank Harary, *Properties of almost all graphs and complexes*, J. Graph Theory 3 (1979) 225–240.
4. Béla Bollobás, Random graphs, Academic Press, 1985.
5. Heinz-Dieter Ebbinghaus and Jörg Flum, Finite Model Theory, Springer-Verlag (1995).
6. Yuri Gurevich, *Evolving algebras 1993: Lipari guide*, in Specification and Validation Methods, ed. E. Börger, Oxford University Press (1995) pp. 9–36. See also the *May 1997 draft of the ASM guide*, Tech Report CSE-TR-336-97, EECS Dept., Univ. of Michigan, 1997. Found at `http://www.eecs.umich.edu/gasm/`.
7. Saharon Shelah, *Choiceless polynomial time logic: inability to express* [paper number 634], these proceedings.

Composition and Submachine Concepts for Sequential ASMs

Egon Börger[1] and Joachim Schmid[2]

[1] Università di Pisa, Dipartimento di Informatica, I-56125 Pisa, Italy
boerger@di.unipi.it (Visiting Microsoft Research, Redmond)
[2] Siemens AG, Corporate Technology, D-81730 Munich, Germany
joachim.schmid@mchp.siemens.de

Abstract. We define three composition and structuring concepts which reflect frequently used refinements of ASMs and integrate standard structuring constructs into the global state based parallel ASM view of computations. First we provide an operator which combines the atomic update view of ASMs with *sequential* machine *execution* and naturally incorporates classical *iteration* constructs into ASMs. For structuring large machines we define their *parameterization*, leading to a notion of possibly recursive submachine calls which sticks to the bare logical minimum needed for sequential ASMs, namely consistency of simultaneous machine operations. For encapsulation and state hiding we provide ASMs with *local state*, *return values* and *error handling*.
Some of these structuring constructs have been implemented in ASM-Gofer. We provide also a proof-theoretic definition which supports the use of common structured proof principles for proving properties for complex machines in terms of properties of their components.

1 Introduction

It has often been observed that Gurevich's definition of Abstract State Machines (ASMs) [13] uses only conditional assignments and supports none of the classical control or data structures. On the one side this leaves the freedom – necessary for *high-level* system design and analysis – to introduce during the modeling process any control or data structure whatsoever which may turn out to be suitable for the application under study. On the other hand it forces the designer to specify standard structures over and over again when they are needed, at the latest when it comes to implement the specification. In this respect ASMs are similar to Abrial's Abstract Machines [1] which are expressed by non-executable pseudo-code without sequencing or loop (Abstract Machine Notation, AMN). In particular there is no notion of submachine and no calling mechanism. For both Gurevich's ASMs and Abrial's Abstract Machines, various notions of refinement have been used to introduce the classical control and data structures. See for example the definition in [15] of recursion as a distributed ASM computation (where calling a recursive procedure is modeled by creating a new instance of multiple agents executing the program for the procedure body) and the definition

P. Clote and H. Schwichtenberg (Eds.): CSL 2000, LNCS 1862, pp. 41–60, 2000.

in [1, 12.5] of recursive AMN calls of an operation as calls to the operation of importing the implementing machine.

Operations of B-Machines [1] and of ASMs come in the form of atomic actions. The semantics of ASMs provided in [13] is defined in terms of a function *next* from states (structures) to states which reflects one step of machine execution. We extend this definition to a function describing, as one step, the result of executing an a priori unlimited number n of basic machine steps. Since n could go to ∞, this naturally leads to consider also non halting computations. We adapt this definition to the view of simultaneous atomic updates in a global state, which is characteristic for the semantics of ASMs, and avoid prescribing any specific syntactic form of encapsulation or state hiding. This allows us to integrate the classical control constructs for *sequentialization and iteration* into the global state based ASM view of computations. Moreover this can be done in a compositional way, supporting the corresponding well known structured proof principles for proving properties for complex machines in terms of properties of their components. We illustrate this by providing structured ASMs for computing arbitrary computable functions, in a way which combines the advantages of functional and of imperative programming. The atomicity of the ASM iteration constructor we define below turned out to be the key for a rigorous definition of the semantics of event triggered exiting from compound actions of UML activity and state machine diagrams, where the intended instantaneous effect of exiting has to be combined with the request to exit nested diagrams sequentially following the subdiagram order, see [5,6].

For structuring large ASMs extensive use has been made of macros as notational shorthands. We enhance this use here by defining the semantics of *named parameterized ASM rules* which include also recursive ASMs. Aiming at a foundation which supports the practitioners' procedural understanding and use of submachine calls, we follow the spirit of the basic ASM concept [13] where domain theoretic complications – arising when explaining what it means to iterate the execution of a machine "until ..." – have been avoided, namely by defining only the one-step computation relation and by relegating fixpoint ("termination") concerns to the metatheory. Therefore we define the semantics of submachine calls only for the case that the possible chain of nested calls of that machine is finite. We are thus led to a notion of calling submachines which mimics the standard imperative calling mechanism and can be used for a definition of recursion in terms of *sequential* (not distributed) ASMs. This definition suffices to justify the submachines used in [8] for a hierarchical decomposition of the Java Virtual Machine into loading, verifying and executing machines for the five principal language layers (imperative core, static classes, object oriented features, exception handling and concurrency).

The third kind of structuring mechanism for ASMs we consider in this paper is of syntactical nature, dealing essentially with name spaces. Parnas' [17] information hiding principle is strongly supported by the ASM concept of external functions which provides also a powerful interface mechanism (see [4]). A more syntax oriented form of information hiding can be naturally incorporated into

ASMs through the notion of *local machine state*, of machines with *return values* and of *error handling* machines which we introduce in Section 5.

Some of these concepts have been implemented in ASMGofer [18], allowing us to define executable versions of the machines for Java and the JVM in [8].

2 Standard ASMs

We start from the definition of basic sequential (i.e. non distributed) ASMs in [13] and survey in this section our notation.

Basic ASMs are built up from *function updates* and *skip* by *parallel composition* and constructs for *if then else*, *let* and *forall*. We consider the *choose*-construct as a special notation for using choice functions, a special class of external functions. Therefore we do not list it as an independent construct in the syntactical definition of ASMs. It appears however in the appendix because the non-deterministic selection of the *choose*-value is directly related to the non-deterministic application of the corresponding deduction rule.

The interpretation of an ASM in a given state $\mathfrak{A}$ depends on the given environment *Env*, i.e. the interpretation $\zeta \in Env$ of its free variables. We use the standard interpretation $[\![t]\!]^{\mathfrak{A}}_{\zeta}$ of terms t in state $\mathfrak{A}$ under variable interpretation ζ, but we often suppress mentioning the underlying interpretation of variables. The semantics of standard ASMs is defined in [13] by assigning to each rule R, given a state $\mathfrak{A}$ and a variable interpretation ζ, an update set $[\![R]\!]^{\mathfrak{A}}_{\zeta}$ which – if consistent – is fired in state $\mathfrak{A}$ and produces the next state $next_R(\mathfrak{A}, \zeta)$.

An update set is a set of *updates*, i.e. a set of pairs (loc, val) where *loc* is a location and *val* is an element in the domain of $\mathfrak{A}$ to which the location is intended to be updated. A location is n-ary function name f with a sequence of length n of elements in the domain of $\mathfrak{A}$, denoted by $f\langle a_1, \ldots, a_n\rangle$. If u is an update set then $Locs(u)$ denotes the set of locations occurring in elements of u ($Locs(u) = \{loc \mid \exists val : (loc, val) \in u\}$). An update set u is called *inconsistent* if u contains at least two pairs (loc, v_1) and (loc, v_2) with $v_1 \neq v_2$ (i.e. $|u| > |Locs(u)|$), otherwise it is called *consistent*.

For a consistent update set u and a state $\mathfrak{A}$, the state $fire_{\mathfrak{A}}(u)$, resulting from firing u in $\mathfrak{A}$, is defined as state $\mathfrak{A}'$ which coincides with $\mathfrak{A}$ except $f^{\mathfrak{A}'}(a) = val$ for each $(f\langle a\rangle, val) \in u$. Firing an inconsistent update set is not allowed, i.e. $fire_{\mathfrak{A}}(u)$ is not defined for inconsistent u. This definition yields the following (partial) next state function $next_R$ which describes *one* application of R in a state with a given environment function $\zeta \in Env$. We often write also $next(R)$ instead of $next_R$.

$$next_R : State(\Sigma) \times Env \to State(\Sigma)$$
$$next_R(\mathfrak{A}, \zeta) = fire_{\mathfrak{A}}([\![R]\!]^{\mathfrak{A}}_{\zeta})$$

The following definitions describe the meaning of standard ASMs. We use R and S for rules, x for variables, s and t for expressions, p for predicates (boolean expressions), and u, v for semantical values and update sets. We write $f^{\mathfrak{A}}$ for

the interpretation of the function f in state $\mathfrak{A}$ and $\zeta' = \zeta\frac{x}{u}$ is the variable environment which coincides with ζ except for x where $\zeta'(x) = u$.

$$
\begin{array}{ll}
[\![x]\!]^{\mathfrak{A}}_{\zeta} & = \zeta(x) \\
[\![f(t_1,\ldots,t_n)]\!]^{\mathfrak{A}}_{\zeta} & = f^{\mathfrak{A}}([\![t_1]\!]^{\mathfrak{A}}_{\zeta},\ldots,[\![t_n]\!]^{\mathfrak{A}}_{\zeta}) \\
[\![\mathbf{skip}]\!]^{\mathfrak{A}}_{\zeta} & = \emptyset \\
[\![f(t_1,\ldots,t_n) := s]\!]^{\mathfrak{A}}_{\zeta} & = \{(f\langle [\![t_1]\!]^{\mathfrak{A}}_{\zeta},\ldots,[\![t_n]\!]^{\mathfrak{A}}_{\zeta}\rangle, [\![s]\!]^{\mathfrak{A}}_{\zeta})\} \\
[\![\{R_1,\ldots,R_n\}]\!]^{\mathfrak{A}}_{\zeta} & = [\![R_1]\!]^{\mathfrak{A}}_{\zeta} \cup \cdots \cup [\![R_n]\!]^{\mathfrak{A}}_{\zeta} \\
[\![\mathbf{if}\ t\ \mathbf{then}\ R\ \mathbf{else}\ S]\!]^{\mathfrak{A}}_{\zeta} & = \begin{cases} [\![R]\!]^{\mathfrak{A}}_{\zeta}, & \text{if } [\![t]\!]^{\mathfrak{A}}_{\zeta} = \text{true}^{\mathfrak{A}} \\ [\![S]\!]^{\mathfrak{A}}_{\zeta}, & \mathbf{otherwise} \end{cases} \\
[\![\mathbf{let}\ x = t\ \mathbf{in}\ R]\!]^{\mathfrak{A}}_{\zeta} & = [\![R]\!]^{\mathfrak{A}}_{\zeta\frac{x}{v}} \text{ where } v = [\![t]\!]^{\mathfrak{A}}_{\zeta} \\
[\![\mathbf{forall}\ x\ \mathbf{with}\ p\ \mathbf{do}\ R]\!]^{\mathfrak{A}}_{\zeta} & = \bigcup_{v\in V} [\![R]\!]^{\mathfrak{A}}_{\zeta\frac{x}{v}} \text{ where } V = \{v \mid [\![p]\!]^{\mathfrak{A}}_{\zeta\frac{x}{v}} = \text{true}^{\mathfrak{A}}\}
\end{array}
$$

Remark: Usually the parallel composition $\{R_1,\ldots,R_n\}$ of rules R_i is denoted by displaying the R_i vertically one above the other.

For a standard ASM R, the update set $[\![R]\!]^{\mathfrak{A}}_{\zeta}$ is defined for any state $\mathfrak{A}$ and for any variable environment ζ, but $next_R(\mathfrak{A},\zeta)$ is undefined if $[\![R]\!]^{\mathfrak{A}}_{\zeta}$ is inconsistent.

3 Sequential Composition and Iteration

The basic composition of ASMs is parallel composition, and this is so for a fundamental reason explained in [14]. It is for practical purposes that in this section we incorporate into ASMs their sequential composition and their iteration, but in a way which fits the basic paradigm of parallel execution of all the rules of a given ASM. The idea is to treat the sequential execution P **seq** Q of two rules P and Q as an "atomic" action, in the same way as executing a function update $f(t_1,\ldots,t_n) := s$, and similarly for the iteration **iterate**(R) of rule R, i.e. the repeated application of sequential composition of R with itself, as long as possible. The notion of repetition yields a definition of the traditional **while** $(cond)$ R construct which is similar to its proof theoretic counterpart in [1, 9.2.1]. Whereas Abrial explicitly excludes sequencing and loop from the specification of abstract machines [1, pg. 373], we take a more pragmatic approach and define them in such a way that they can be used coherently in two ways, depending on what is needed, namely to provide black-box descriptions of abstract submachines or glass-box views of their implementation (refinement).

3.1 Sequence Constructor

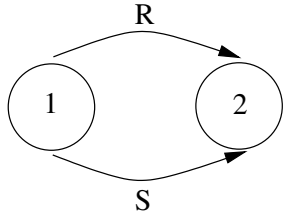
If one wants to specify executing one standard ASM after another, this has to be explicitly programmed. Consider for example the function *pop_back* in the Standard Template Library for C++ (abstracting from concrete data structures). The function deletes the last element in a list. Assume further that we have already defined rules *move_last* and *delete* where *move_last* sets the list pointer

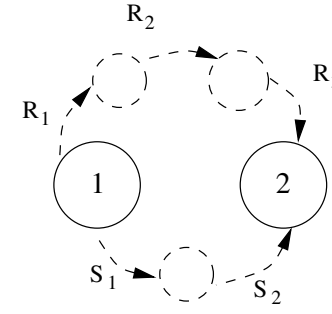
to the last element and *delete* removes the current element. One may be tempted to program *pop_back* as follows to first execute *move_last* and then *delete*:

pop_back ≡
  if *mode* = *Move* **then**
    move_last
    mode := *Delete*
  if *mode* = *Delete*
    delete
    mode := *Move*

This definition has the drawback that the user of *pop_back* must know that the action to be completed needs two steps, which really is an implementation feature. Moreover the dynamic function *mode*, which is used to *program* the sequential ordering, is supposed to be initialized by *Move*. Such an explicit programming of execution order quickly becomes a stumbling block for large specifications, in particular the initialization is not easily guaranteed without introducing an explicit initialization mechanism.

Another complication arises when sequentialized rules are used to refine abstract machines. In the machine on the left side of the picture below, assume that the simultaneous execution of the two rules R and S in state 1 leads to state 2. The machine on the right side is supposed to refine the machine on the left side with rules R and S refined into the sequence of rules $R_1R_2R_3$ and S_1S_2 respectively. There is no obvious general scheme to interleave the R_i-rules and the S_j-rules, using a *mode* function as above. What should happen if rule R_2 modifies some locations which are read by S_2? In such cases R and S could not be refined independently of each other.





Therefore we introduce a sequence constructor yielding a rule P **seq** Q which can be inserted into another ASM but whose semantical effect is nevertheless the sequential execution of the two rules P and Q. If the new rule P **seq** Q has to share the same status as any other ASM rule together with which it may be executed in parallel, one can define the execution of P **seq** Q only as an atomic action. Obviously this is only a way to "view" the sequential machine from outside; its refined view reveals its internal structure and behavior, constituted by the non atomic execution, namely in two steps, of first P and then Q.

Syntactically the sequential composition P **seq** Q of two rules P and Q is defined to be a rule. The semantics is defined as first executing P, obtaining an intermediate state, followed by executing Q in the intermediate state. This is formalized by the following definition of the update set of P **seq** Q in state $\mathfrak{A}$.

Semantics: Let P and Q be rules. We define

$$[\![P \textbf{ seq } Q]\!]^{\mathfrak{A}} = [\![P]\!]^{\mathfrak{A}} \oplus [\![Q]\!]^{\mathfrak{A}'}$$

where $\mathfrak{A}' = next_P(\mathfrak{A})$ is the state obtained by firing the update set of P in state $\mathfrak{A}$, if this is defined; otherwise $\mathfrak{A}'$ can be chosen arbitrarily. The operator $\oplus$ denotes the merging for update sets.

The merging of two update sets u and v by the operator $\oplus$ reflects that an update in v overwrites an update in u if it is for the same location, since through a destructive assignment $s := t$ the previous value of s is lost. We merge an update set v with u (i.e. $u \oplus v$) only if u is consistent, otherwise we stick to u because then we want both $fire_{\mathfrak{A}}(u)$ and $fire_{\mathfrak{A}}(u \oplus v)$ to be undefined.

$$u \oplus v = \begin{cases} \{(loc, val) \mid (loc, val) \in u \wedge loc \notin Locs(v)\} \cup v, & consistent(u) \\ u, & \textbf{otherwise} \end{cases}$$

Proposition 1. (Persistence of inconsistency)

If $[\![P]\!]^{\mathfrak{A}}$ is not consistent, then $[\![P \textbf{ seq } Q]\!]^{\mathfrak{A}} = [\![P]\!]^{\mathfrak{A}}$

The next proposition shows that the above definition of the **seq** constructor captures the intended classical meaning of sequential composition of machines, if we look at them as state transforming functions[1]. Indeed we could have defined **seq** via the composition of algebra transforming functions, similarly to its axiomatically defined counterpart in Abrial's AMN [1] where **seq** comes as concatenation of generalized substitutions.

Proposition 2. (Compositionality of **seq**)

$$next(P \textbf{ seq } Q) = next(Q) \circ next(P)$$

This characterization illustrates that **seq** has the expected semiring properties on update sets.

Proposition 3. The ASM constructor **seq** has a left and a right neutral element and is associative, i.e. for rules P, Q, and R the following holds:

$$[\![\textbf{skip seq } R]\!]^{\mathfrak{A}} = [\![R \textbf{ seq skip}]\!]^{\mathfrak{A}} = [\![R]\!]^{\mathfrak{A}}$$

$$[\![P \textbf{ seq } (Q \textbf{ seq } R)]\!]^{\mathfrak{A}} = [\![(P \textbf{ seq } Q) \textbf{ seq } R]\!]^{\mathfrak{A}}$$

[1] We assume that $f(x)$ is undefined if x is undefined, for every function f (f is strict).

3.2 Iteration Constructor

Once a sequence operator is defined, one can apply it repeatedly to define the iteration of a rule. This provides a natural way to define for ASMs an iteration construct which encapsulates a computation with a finite but a priori not explicitly known number of iterated steps into an atomic action (one-step computation). As a by-product we obtain the classical loop and while constructs, cf. [1, 9.2].

The intention of rule iteration is to execute the given rule again and again – as long as *needed* and as long as *possible*. We define

$$R^n = \begin{cases} \textbf{skip}, & n = 0 \\ R^{n-1} \textbf{ seq } R, & n > \end{cases}$$

Denote by $\mathfrak{A}_n$ the state obtained by firing the update set of the rule R^n in state $\mathfrak{A}$, if defined (i.e. $\mathfrak{A}_n = next_{R^n}(\mathfrak{A})$).

There are two natural stop situations for iterated ASM rule application, namely when the update set becomes empty (the case of successful termination) and when it becomes inconsistent (the case of failure, given the persistence of inconsistency as formulated in Proposition 1).[2] Both cases provide a fixpoint $\lim_{n\to\infty} [\![R^n]\!]^{\mathfrak{A}}$ for the sequence $([\![R^n]\!]^{\mathfrak{A}})_{n>0}$ which becomes constant if a number n is found where the update set of R, in the state obtained by firing R^{n-1}, is empty or inconsistent.

Proposition 4. (Fixpoint Condition)

$\forall m \geq n > 0$ the following holds:
if $[\![R]\!]^{\mathfrak{A}_{n-1}}$ is not consistent or if it is empty, then $[\![R^m]\!]^{\mathfrak{A}} = [\![R^n]\!]^{\mathfrak{A}}$

Therefore we extend the syntax of ASM rules by **iterate**(R) to denote the iteration of rule R and define its semantics as follows.

Semantics: Let R be a rule. We define

$$[\![\textbf{iterate}(R)]\!]^{\mathfrak{A}} = \lim_{n\to\infty} [\![R^n]\!]^{\mathfrak{A}}, \quad \text{if } \exists\, n \geq 0 : [\![R]\!]^{\mathfrak{A}_n} = \emptyset \vee \neg \mathit{consistent}([\![R]\!]^{\mathfrak{A}_n})$$

The sequence $([\![R^n]\!]^{\mathfrak{A}})_{n>0}$ eventually becomes constant only upon termination or failure. Otherwise the computation diverges and the update set for the iteration is undefined. An example for a machine R which naturally produces a diverging (though in other contexts useful) computation is **iterate**$(a := a + 1)$, see [16, Exl. 2, pg. 350].

[2] We do not include here the case of an update set whose firing does not change the given state, although including this case would provide an alternative stop criterion which is also viable for implementations of ASMs.

Example 1. (Usage of **iterate** for starting the Java class initialization process)

The ASM model for Java in [9] includes the initialization of classes which in Java is done implicitly at the first use of a class. Since the Java specification requires that the superclass of a class c is initialized before c, the starting of the class initialization is iterated until an initialized class c' is encountered (i.e. satisfying $initialized(c')$, as eventually will happen towards the top of the class hierarchy). We define the initialization of class *class* as follows:

$$
\begin{array}{l}
initialize \equiv \\
\quad c := class \ \textbf{seq iterate}(\textbf{if } \neg initialized(c) \textbf{ then} \\
\qquad\qquad\qquad\qquad\qquad\quad createInitFrame(c) \\
\qquad\qquad\qquad\qquad\quad \textbf{if } \neg initialized(superClass(c)) \textbf{ then} \\
\qquad\qquad\qquad\qquad\qquad\quad c := superClass(c))
\end{array}
$$

The finiteness of the acyclic class hierarchy in Java guarantees that this rule yields a well defined update set. The rule abstracts from the standard sequential implementation (where obviously the class initialization is started in a number of steps depending on how many super classes the given class has which are not yet initialized) and offers an atomic operation to push all initialization methods in the right order onto the frame stack.

The macro to create new initialization frames can be defined as follows. The current computation state, consisting of *method*, *program*, program position *pos* and *localVars*, is pushed onto the *frames* stack and is updated for starting the initialization method of the given *class* at position 0 with empty local variables set.

$$
\begin{array}{l}
createInitFrame(c) \equiv \\
\quad classState(c) := InProgress \\
\quad frames \quad := frames \cdot (method, program, pos, localVars) \\
\quad method \quad := c/\texttt{<clinit>} \\
\quad program \quad := body(c/\texttt{<clinit>}) \\
\quad pos \quad := 0 \\
\quad localVars \quad := \emptyset
\end{array}
$$

While and Until. The iteration yields a natural definition of classical loop and while constructs. A *while loop* repeats the execution of the *while body* as long as a certain condition holds.

$$\textbf{while } (cond)\ R = \textbf{iterate}(\textbf{if } cond \textbf{ then } R)$$

This *while* loop, if started in state $\mathfrak{A}$, terminates if eventually $[\![R]\!]^{\mathfrak{A}_n}$ becomes empty or the condition *cond* becomes *false* in $\mathfrak{A}_n$ (with consistent and non empty previous update sets $[\![R]\!]^{\mathfrak{A}_i}$ and previous states $\mathfrak{A}_i$ satisfying *cond*). If the iteration of R reaches an inconsistent update set (failure) or yields an infinite sequence of consistent non empty update sets, then the state resulting from executing the while loop starting in $\mathfrak{A}$ is not defined (divergence of the while

loop). Note that the function $next(\textbf{while}\ (cond)\, R)$ is undefined in these two cases on $\mathfrak{A}$.

A *while* loop may satisfy more than one of the above conditions, like **while** (*false*) **skip**. The following examples illustrate the typical four cases:

- (success) **while** (*cond*) **skip**
- (success) **while** (*false*) R
- (failure) **while** (*true*) $a := 1$
  $a := 2$
- (divergence) **while** (*true*) $a := a$

Example 2. (Usage of **while**)

The following iterative ASM defines a while loop to compute the factorial function for given argument x and stores the result in a location *fac*. It uses multiplication as given (static) function. We will generalize this example in the next section to an ASM analogue to the Böhm-Jacopini theorem on structured programming [3].

$$compute\_fac \equiv (fac := 1)\ \textbf{seq}\ (\textbf{while}\ (x > 0)\ fac := x * fac \quad x := x - 1)$$

Remark: As usual one can define the *until loop* in terms of **while** and **seq** as first executing the body once and then behaving like a while loop:

$$\textbf{do}\ R\ \textbf{until}\ (cond) = R\ \textbf{seq}\ (\textbf{while}\ (\neg cond)\, R).$$

The sequencing and iteration concepts above apply in particular to the *Mealy-ASMs* defined in [4] for which they provide the sequencing and the feedback operators. The fundamental parallel composition of ASMs provides the concept of parallel composition of Mealy automata for free. These three constructs allow one to apply to Mealy-ASMs the decomposition theory which has been developed for finite state machines in [10].

3.3 Böhm-Jacopini ASMs

The sequential and iterative composition of ASMs yields a class of machines which are known from [3] to be appropriate for the computation of partial recursive functions. We illustrate in this section how these *Böhm-Jacopini-ASMs* naturally combine the advantages of the Gödel-Herbrand style functional definition of computable functions and of the Turing style imperative description of their computation.

Let us call Böhm-Jacopini-ASM any ASM which can be defined, using the sequencing and the iterator constructs, from basic ASMs whose functions are restricted as defined below to input, output, controlled functions and some simple static functions. For each Böhm-Jacopini-ASM M we allow only one external function, a 0-ary function for which we write in_M. The purpose of this function

is to contain the number sequence which is given as input for the computation of the machine. Similarly we write out_M for the unique (0-ary) function which will be used to receive the output of M. Adhering to the usual practice one may also require that the M-output function appears only on the left hand side of M-updates, so that it does not influence the M-computation and is not influenced by the environment of M. As static functions we admit only the *initial* functions of recursion theory, i.e. the following functions from Cartesian products of natural numbers into the set $\mathbb{N}$ of natural numbers: $+1$, all the projection functions U_i^n, all the constant functions C_i^n and the characteristic function of the predicate $\neq 0$.

Following the standard definition we call a number theoretic function $f : \mathbb{N}^n \to \mathbb{N}$ computable by an ASM M if for every n-tuple $x \in \mathbb{N}^n$ of arguments on which f is defined, the machine started with input x terminates with output $f(x)$. By "M started with input x" we mean that M is started in the state where all the dynamic functions different from in_M are completely undefined and where $in_M = x$. Assuming the external function in_M not to change its value during an M-computation, it is natural to say that M terminates in a state with output y, if in this state out_M gets updated for the first time, namely to y.

Proposition 5. (Structured Programming Theorem)

Every partial recursive function can be computed by a Böhm-Jacopini-ASM.

Proof. We define by induction for each partial recursive function f a machine F computing it. Each initial function f of recursion theory is computed by the following machine F consisting of only one function update which reflects the defining equation of f.

$$F \equiv out_F := f(in_F)$$

For the inductive step it suffices to construct, for any partial recursive definition of a function f from its constituent functions f_i, a machine F which mimics the standard evaluation procedure underlying that definition. We define the following macros for using a machine F for given arguments in, possibly including to assign its output to a location out:

$$\begin{aligned} F(in) \quad &\equiv in_F := in \textbf{ seq } F \\ out := F(in) &\equiv F(in) \textbf{ seq } out := out_F \end{aligned}$$

We start with the case of function composition. If functions $g, h_1, \ldots, h_m$ are computed by Böhm-Jacopini-ASMs $G, H_1, \ldots, H_m$, then their composition f defined by $f(x) = g(h_1(x), \ldots, h_m(x))$ is computed by the following machine[3] F:

$$F \equiv \{H_1(in_F), \ldots, H_m(in_F)\} \textbf{ seq } out_F := G(out_{H_1}, \ldots, out_{H_m})$$

[3] For reasons of simplicity but without loss of generality we assume that the submachines have pairwise disjoint signatures.

Unfolding this structured program reflects the order one *has* to follow for evaluating the subterms in the defining equation for f, an order which is implicitly assumed in the equational (functional) definition. First the input is passed to the constituent functions h_i to compute their values, whereby the input functions of H_i become controlled functions of F. The parallel composition of the submachines $H_i(in_F)$ reflects that any order is allowed here. Then the sequence of out_{H_i} is passed as input to the constituent function g. Finally g's value on this input is computed and assigned as output to out_F.

Similarly let a function f be defined from g, h by primitive recursion:

$$f(x,0) = g(x), \quad f(x,y+1) = h(x,y,f(x,y))$$

and let Böhm-Jacopini-ASMs G, H be given which compute g, h. Then the following machine F computes f, composed as sequence of three submachines. The start submachine of F evaluates the first defining equation for f by initializing the recursor *rec* to 0 and the intermediate value *ival* to $g(x)$. The *while* submachine evaluates the second defining equation for f for increased values of the recursor as long as the input value y has not been reached. The output submachine provides the final value of *ival* as output.

$$\begin{aligned} F \equiv\ & \textbf{let } (x,y) = in_F \textbf{ in} \\ & \{ival := G(x),\ rec := 0\} \textbf{ seq} \\ & (\textbf{while } (rec < y)\, \{ival := H(x, rec, ival),\ rec := rec + 1\}) \textbf{ seq} \\ & out_F := ival \end{aligned}$$

If f is defined from g by the μ-operator, i.e. $f(x) = \mu y(g(x,y) = 0)$, and if a Böhm-Jacopini-ASM G computing g is given, then the following machine F computes f. The start submachine computes $g(x, rec)$ for the initial recursor value 0, the iterating machine computes $g(x, rec)$ for increased values of the recursor until 0 shows up as computed value of g, in which case the reached recursor value is set as output.

$$\begin{aligned} F \equiv\ & \{G(in_F, 0),\ rec := 0\} \textbf{ seq} \\ & (\textbf{while } (out_G \neq 0)\, \{G(in_F, rec + 1),\ rec := rec + 1\}) \textbf{ seq} \\ & out_F := rec \end{aligned}$$

Remark. The construction of Böhm-Jacopini-ASMs illustrates, through the idealized example of computing recursive functions, how ASMs allow to pragmatically reconcile the often discussed conceptual dichotomy between functional and imperative programming. In the context of discussing the "functional programming language" Gödel used to exhibit undecidable propositions in Principia Mathematica, as opposed to the "imperative programming language" developed by Turing and used in his proof of the unsolvability of the *Entscheidungsproblem* (see [7]), Martin Davis [12] states:

> "The programming languages that are mainly in use in the software industry (like C and FORTRAN) are usually described as being *imperative*. This is because the successive lines of programs written in these

> languages can be thought of as *commands* to be executed by the computer ... In the so-called *functional* programming languages (like LISP) the lines of a program are definitions of operations. Rather than telling the computer what to do, they *define* what it is that the computer is to provide."

The equations which appear in the Gödel-Herbrand type equational definition of partial recursive functions "define what it is that the computer is to provide" only within the environment for evaluation of subterms. The corresponding Böhm-Jacopini-ASMs constructed above make this context explicit, exhibiting how to evaluate the subterms when using the equations (updates), as much as needed to make the functional shorthand work correctly. We show in the next section how this use of shorthands for calling submachines, which appear here only in the limited context of structured WHILE programs, can be generalized as to make it practical without loss of rigor.

4 Parameterized Machines

For structuring large ASMs extensive use has been made of macros which, semantically speaking, are mere notational shorthands, to be substituted by the body of their definition. We enhance this use here by introducing named parameterized ASM rules which in contrast to macros also support recursive ASMs.

We provide a foundation which justifies the application of named parameterized ASMs in a way which supports the practitioners' procedural understanding. Instead of guaranteeing within the theory, typically through a fixpoint operator, that under certain conditions iterated calls of recursive rules yield as "result" a first-class mathematical "object" (namely the fixpoint), we take inspiration from the way Kleene proved his recursion theorem [16, Section 66] and leave it to the programmer to guarantee that a possibly infinite chain of recursive procedure calls is indeed well founded with respect to some partial order.

We want to allow a named parameterized rule to be used in the same way as all other rules. For example, if f is a function with arity 1 and R is a named rule expecting two parameters, then $R(f(1), 2)$ should be a legitimate rule, too. In particular we want to allow rules as parameters, like in the following example where the given dynamic function *stdout* is updated to "hello world":

rule $R(output) =$
  $output$(`"hello world"`)

rule $output\_to\_stdout(msg)$
  $stdout := msg$

$R(output\_to\_stdout)$

Therefore we extend the inductive syntactic definition for rules by the following new clause, called a rule application with actual parameters $a_1, \ldots, a_n$:

$R(a_1, \ldots, a_n)$

and coming with a rule definition of the following form:

rule $R(x_1, \ldots, x_n) = body$

where *body* is a rule. R is called the rule name, $x_1, \ldots, x_n$ are the formal parameters of the rule definition. They bind the free occurrences of the variables $x_1, \ldots, x_n$ in *body*.

The basic intuition the practice of computing provides for the interpretation of a named rule is to define its semantics as the interpretation of the rule body with the formal parameters replaced by the actual arguments. In other words we unfold nested calls of a recursive rule R into a sequence $R_1, R_2, \ldots$ of rule incarnations where each R_i may trigger one more execution of the rule body, relegating the interpretation of possibly yet another call of R to the next incarnation R_{i+1}. This may produce an infinite sequence, namely if there is no ordering of the procedure calls with respect to which the sequence will decrease and reach a basis for the recursion. In this case the semantics of the call of R is undefined. If however a basis for the recursion does exist, say R_n, it yields a well defined value for the semantics of R through the chain of successive calls of R_i; namely for each $0 \leq i < n$ with $R = R_0$, R_i inherits its semantics from R_{i+1}.

Semantics: Let R be a named rule declared by **rule** $R(x_1, \ldots, x_n) = body$, let $\mathfrak{A}$ be a state.

If $[\![body[a_1/x_1, \ldots, a_n/x_n]]\!]^{\mathfrak{A}}$ is defined, then
$[\![R(a_1, \ldots, a_n)]\!]^{\mathfrak{A}} = [\![body[a_1/x_1, \ldots, a_n/x_n]]\!]^{\mathfrak{A}}$

For the rule definition **rule** $R(x) = R(x)$ this interpretation yields no value for any $[\![R(a)]\!]^{\mathfrak{A}}$, see [16, Example 1, page 350]. In the following example the update set for $R(x)$ is defined for all $x \leq 10$, with the empty set as update set, and is not defined for any $x > 10$.

rule $R(x) =$ **if** $x < 10$ **then** $R(x + 1)$
if $x = 10$ **then skip**
if $x > 10$ **then** $R(x + 1)$

Example 3. (Defining **while** by a named rule)

Named rules allow us to define the *while loop* recursively instead of iteratively:

rule *while*$(cond, R) =$
if *cond* **then**
R **seq** *while*$(cond, R)$

This recursively defined **while** operator behaves differently from the iteratively defined **while** of the preceding section in that it leads to termination only if the condition *cond* will become eventually false, and not in the case that eventually the update set of R becomes empty. For example the semantics of the recursively defined *while*$(true,$ **skip**$)$ is not defined.

Example 4. (Starting Java class initialization)

We can define the Java class initialization of Example 1 also in terms of a recursive named rule, avoiding the local input variable to which the actual parameter is assigned at the beginning.

rule *initialize*(c) =
  if *initialized*(*superClass*(c)) **then**
    createInitFrame(c)
  else
    createInitFrame(c) **seq** *initialize*(*superClass*(c))

Remark: Iterated execution of (sub)machines R, started in state $\mathfrak{A}$, unavoidably leads to possibly undefined update sets $[\![R]\!]^{\mathfrak{A}}$. As a consequence $[\![R]\!]^{\mathfrak{A}} = [\![S]\!]^{\mathfrak{A}}$ denotes that either both sides of the equation are undefined or both are defined and indeed have the same value. In the definitions above we adhered to an algorithmic definition of $[\![R]\!]^{\mathfrak{A}}$, namely by computing its value from the computed values $[\![S]\!]^{\mathfrak{A}}$ of the submachines S of R. In the appendix we give a deduction calculus for proving statements $[\![R]\!]^{\mathfrak{A}} = u$ meaning that $[\![R]\!]^{\mathfrak{A}}$ is defined and has value u.

5 Further Concepts

In this section we enrich named rules with a notion of local state, show how parameterized ASMs can be used as machines with return value, and introduce error handling for ASMs which is an abstraction of exception handling as found in modern programming languages.

5.1 Local State

Basic ASMs come with a notion of state in which all the dynamic functions are global. The use of only locally visible parts of the state, like variables declared in a class, can naturally be incorporated into named ASMs. It suffices to extend the definition of named rules by allowing some dynamic functions to be declared as local, meaning that each call of the rule works with its own incarnation of local dynamic functions f which are to be initialized upon rule invocation by an initialization rule $Init(f)$. Syntactically we allow definitions of named rules of the following form:

rule *name*$(x_1, \ldots, x_n)$ =
  local $f_1[Init_1]$
  $\vdots$
  local $f_k[Init_k]$
  body

where *body* and $Init_i$ are rules. The formal parameters $x_1, \ldots, x_n$ bind the free occurrences of the corresponding variables in *body* and $Init_i$. The functions

$f_1, \ldots, f_k$ are treated as local functions whose scope is the rule where they are introduced. They are not part of the signature of the ASM. $Init_i$ is a rule used for the initialization of f_i. We write **local** $f := t$ for **local** $f[f := t]$.

For the semantic interpretation of a call of a rule with local dynamic functions, the updates to the local functions are collected together with all other function updates made through executing the body. This includes the updates required by the initialization rules. The restriction of the scope of the local functions to the rule definition is obtained by then removing from the update set u, which is available after the execution of the body of the call, the set $Updates(f_1, \ldots, f_k)$ of updates concerning the local functions $f_1, \ldots, f_k$. This leads to the following definition.

Semantics: Let R be a rule declaration with local functions as given above. If the right side of the equation is defined, we set:

$$[\![R(a_1, \ldots, a_n)]\!]^{\mathfrak{A}} = \\ [\![(\{Init_1, \ldots, Init_k\} \textbf{ seq } body)[a_1/x_1, \ldots, a_n/x_n]]\!]^{\mathfrak{A}} \setminus Updates(f_1, \ldots, f_k)$$

We assume that there are no name clashes for local functions between different incarnations of the same rule (i.e. each rule incarnation has its own set of local dynamic functions).

Example 5. (Usage of local dynamic functions)

The use of local dynamic functions is illustrated by the following rule computing a function f defined by a primitive recursion from functions g and h which are used here as static functions. The rule mimics the corresponding Böhm-Jacopini machine in Proposition 5.

rule $F(x, y) =$
  local $ival := g(x)$
  local $rec := 0$
  $(\textbf{while } (rec < y) \{ival := h(x, rec, ival),\ rec := rec + 1\})$ **seq**
  $out := ival$

5.2 ASMs with Return Value

In the preceding example, for outputting purposes the value resulting from the computation is stored in a global dynamic function out. This formulation violates good information hiding principles. To store the return value of a rule R in a location which is determined by the rule caller and is independent of R, we use the following notation for a new rule:

$$l \leftarrow R(a_1, \ldots, a_n)$$

where R is a named rule with n parameters in which a 0-ary (say reserved) function $result$ does occur with the intended role to store the return value. Let

rule $R(x_1, \ldots, x_n) = body$ be the declaration for R, then the semantic of $l \leftarrow R(a_1, \ldots, a_n)$ is defined as the semantics of $R_l(a_1, \ldots, a_n)$ where R_l is defined like R with *result* replaced by l:

rule $R_l(x_1, \ldots, x_n) = body[l/result]$

In the definition of the rule R by *body*, the function name *result* plays the role of a placeholder for a location, denoting the interface which is offered for communicating results from any rule execution to its caller. One can apply simultaneously two rules $l \leftarrow R(a_1, \ldots, a_n)$ and $l' \leftarrow R(a'_1, \ldots, a'_n)$ with different return values for l and l'.

Remark: When using $l \leftarrow R(a_1, \ldots, a_n)$ with a term l of form $f(t_1, \ldots, t_n)$, a good encapsulation discipline will take care that R does not modify the values of t_i, because they contribute to determine the location where the caller expects to find the return value.

Example 6. (Using return values)

Using this notation the above Example 5 becomes $f(x, y) \leftarrow F(x, y)$ where moreover one can replace the use of the auxiliary static functions g, h by calls to submachines G, H computing them, namely $ival \leftarrow G(x)$ and $ival \leftarrow H(x, rec, ival)$.

Example 7. (Recursive machine computing the factorial function, using multiplication as static function.)

rule $Fac(n) =$
  local $x := 1$
  if $n = 1$ **then**
    $result := 1$
  else
    $(x \leftarrow Fac(n-1))$ **seq** $result := n * x$

5.3 Error Handling

Programming languages like C++ or Java support exceptions to separate error handling from "normal" execution of code. Producing an inconsistent update set is an abstract form of throwing an exception. We therefore introduce a notion of catching an inconsistent update set and of executing error code.

The semantics of **try** R **catch** $f(t_1, \ldots, t_n)$ S is the update set of R if either this update set is consistent ("normal" execution) or it is inconsistent but the location *loc* determined by $f(t_1, \ldots, t_n)$ is not updated inconsistently. Otherwise it is the update set of S.

Since the rule enclosed by the **try** block is executed either completely or not at all, there is no need for any **finally** clause to remove trash.

Semantics: Let R and S be rules, f a dynamic function with arguments $t_1, \ldots, t_n$. We define

$$[\![\textbf{try } R \textbf{ catch } f(t_1, \ldots, t_n)\ S]\!]^{\mathfrak{A}} = \begin{cases} v, & \exists\, v_1 \neq v_2 : (loc, v_1) \in u \wedge (loc, v_2) \in u \\ u, & \textbf{otherwise} \end{cases}$$

where $u = [\![R]\!]^{\mathfrak{A}}$ and $v = [\![S]\!]^{\mathfrak{A}}$ are the update sets of R and S respectively, and loc is the location $f\langle [\![t_1]\!]^{\mathfrak{A}}, \ldots, [\![t_n]\!]^{\mathfrak{A}} \rangle$.

6 Related Work

The sequence operator defined by Zamulin in [19] differs from our concept for rules leading to inconsistent update sets where it is not associative, due to Zamulin's definition of the merge operator for update sets. For consistent update sets Zamulin's loop constructor coincides with our while definition in Example 2.

In Anlauff's XASM [2], calling an ASM is the iteration of a rule until a certain condition holds. [2] provides no formal definition of this concept, but for consistent update sets the XASM implementation seems to behave like our definition of *iterate*.

Named rules with parameters appear in the ASM Workbench [11] and in XASM [2], but with parameters restricted to terms. The ASM Workbench does not allow recursive rules. Recursive ASMs have also been proposed by Gurevich and Spielmann [15]. Their aim was to justify recursive ASMs within distributed ASMs [13]. If R is a rule executed by agent a and has two recursive calls to R, then a creates two new agents a_1 and a_2 which execute the two corresponding recursive calls. The agent a waits for termination of his slaves a_1 and a_2 and then combines the result of both computations. This is different from our definition where executing a recursive call needs only one step, from the caller's view, so that the justification remains within purely sequential ASMs without invoking concepts from distributed computing. Through our definition the distinction between suspension and reactivation tasks in the iterative implementation of recursion becomes a matter of choosing the black-box or the glass-box view for the recursion. The updates of a recursive call are collected and handed over to the calling machine as a whole to determine the state following in the black-box view the calling state. Only the glass-box view provides a refined inspection of how this collection is computed.

Acknowledgments. For critical comments on earlier versions of this paper[4] we thank Giuseppe Del Castillo, Martin Davis, Jim Huggins, Alexander Knapp, Peter Päppinghaus, Robert Stärk, Margus Vianes, and Alexandre Zamulin.

[4] Presented to the IFIP Working Group 1.3 on Foundations of System Specification, Bonas (France) 13.-15.9.1999, and to the International ASM'2000 Workshop, Ascona (Switzerland) 20.-24.3.2000

A Deduction Rules for Computing Update Sets

The following rules provide a calculus for computing the semantics of standard ASMs and for the constructs introduced in this paper.

We use R, R_i, and S for rules, f for functions, x for variables, s and t for expressions, p for predicates (boolean expressions), and u and v for semantical values and update sets.

Standard ASMs

$$\frac{\forall\, i : [\![t_i]\!]^{\mathfrak{A}}_{\zeta} = v_i}{[\![f(t_1,\ldots,t_n)]\!]^{\mathfrak{A}}_{\zeta} = f^{\mathfrak{A}}(v_1,\ldots,v_n)} \qquad \frac{}{[\![x]\!]^{\mathfrak{A}}_{\zeta} = \zeta(x)}\ \mathit{variable}(x)$$

$$\frac{}{[\![\mathbf{skip}]\!]^{\mathfrak{A}}_{\zeta} = \emptyset} \qquad \frac{[\![t]\!]^{\mathfrak{A}}_{\zeta} = \mathrm{true}^{\mathfrak{A}},\ [\![R]\!]^{\mathfrak{A}}_{\zeta} = u}{[\![\mathbf{if}\ t\ \mathbf{then}\ R\ \mathbf{else}\ S]\!]^{\mathfrak{A}}_{\zeta} = u}$$

$$\frac{\forall\, i : [\![t_i]\!]^{\mathfrak{A}}_{\zeta} = v_i,\ [\![s]\!]^{\mathfrak{A}}_{\zeta} = u}{[\![f(t_1,\ldots,t_n) := s]\!]^{\mathfrak{A}}_{\zeta} = \{(f\langle v_1,\ldots,v_n\rangle, u)\}} \qquad \frac{[\![t]\!]^{\mathfrak{A}}_{\zeta} = \mathrm{false}^{\mathfrak{A}},\ [\![S]\!]^{\mathfrak{A}}_{\zeta} = u}{[\![\mathbf{if}\ t\ \mathbf{then}\ R\ \mathbf{else}\ S]\!]^{\mathfrak{A}}_{\zeta} = u}$$

$$\frac{\forall\, i : [\![R_i]\!]^{\mathfrak{A}}_{\zeta} = u_i}{[\![\{R_1,\ldots,R_n\}]\!]^{\mathfrak{A}}_{\zeta} = u_1 \cup \ldots \cup u_n} \qquad \frac{[\![t]\!]^{\mathfrak{A}}_{\zeta} = v,\ [\![R]\!]^{\mathfrak{A}}_{\zeta\frac{x}{v}} = u}{[\![\mathbf{let}\ x = t\ \mathbf{in}\ R]\!]^{\mathfrak{A}}_{\zeta} = u}$$

$$\frac{V = \{v_1,\ldots,v_n\},\ \forall\, i : [\![R]\!]^{\mathfrak{A}}_{\zeta\frac{x}{v_i}} = u_i}{[\![\mathbf{forall}\ x\ \mathbf{with}\ p\ \mathbf{do}\ R]\!]^{\mathfrak{A}}_{\zeta} = u_1 \cup \ldots \cup u_n}\ V = \{v \mid [\![p]\!]^{\mathfrak{A}}_{\zeta\frac{x}{v}} = \mathrm{true}^{\mathfrak{A}}\}$$

$$\frac{[\![p]\!]^{\mathfrak{A}}_{\zeta\frac{x}{v}} = \mathrm{true}^{\mathfrak{A}},\ [\![R]\!]^{\mathfrak{A}}_{\zeta\frac{x}{v}} = u}{[\![\mathbf{choose}\ x\ \mathbf{with}\ p\ \mathbf{do}\ R]\!]^{\mathfrak{A}}_{\zeta} = u}$$

$$\frac{}{[\![\mathbf{choose}\ x\ \mathbf{with}\ p\ \mathbf{do}\ R]\!]^{\mathfrak{A}}_{\zeta} = \emptyset}\ \nexists\, v : [\![p]\!]^{\mathfrak{A}}_{\zeta\frac{x}{v}} = \mathrm{true}^{\mathfrak{A}}$$

Sequential Composition

$$\frac{[\![R]\!]^{\mathfrak{A}}_{\zeta} = u,\ [\![S]\!]^{\mathit{fire}_{\mathfrak{A}}(u)}_{\zeta} = v}{[\![R\ \mathbf{seq}\ S]\!]^{\mathfrak{A}}_{\zeta} = u \oplus v}\ \mathit{consistent}(u) \qquad \frac{[\![R]\!]^{\mathfrak{A}}_{\zeta} = u}{[\![R\ \mathbf{seq}\ S]\!]^{\mathfrak{A}}_{\zeta} = u}\ \mathit{inconsistent}(u)$$

Iteration

$$\frac{[\![R^n]\!]^{\mathfrak{A}}_{\zeta} = u}{[\![\mathbf{iterate}(R)]\!]^{\mathfrak{A}}_{\zeta} = u}\ n \geq 0, \mathit{inconsistent}(u)$$

$$\frac{[\![R^n]\!]^{\mathfrak{A}}_{\zeta} = u,\ [\![R]\!]^{\mathit{fire}_{\mathfrak{A}}(u)}_{\zeta} = \emptyset}{[\![\mathbf{iterate}(R)]\!]^{\mathfrak{A}}_{\zeta} = u}\ n \geq 0, \mathit{consistent}(u)$$

Parameterized Rules with Local State
Let R be a named rule as in Section 5.1.

$$\frac{[\![(\{Init_1, \ldots, Init_k\} \textbf{ seq } body)[a_1/x_1, \ldots, a_n/x_n]]\!]^{\mathfrak{A}}_{\zeta} = u}{[\![R(a_1, \ldots, a_n)]\!]^{\mathfrak{A}}_{\zeta} = u \setminus Updates(f_1, \ldots, f_k)}$$

Error Handling

$$\frac{[\![R]\!]^{\mathfrak{A}}_{\zeta} = u}{[\![\textbf{try } R \textbf{ catch } f(t_1, \ldots, t_n)\ S]\!]^{\mathfrak{A}}_{\zeta} = u} \quad \begin{array}{l} \nexists\, v_1 \neq v_2 : (loc, v_1) \in u \wedge (loc, v_2) \in u \\ where \;\; loc = f\langle [\![t_1]\!]^{\mathfrak{A}}_{\zeta}, \ldots, [\![t_n]\!]^{\mathfrak{A}}_{\zeta} \rangle \end{array}$$

$$\frac{[\![R]\!]^{\mathfrak{A}}_{\zeta} = u,\ [\![S]\!]^{\mathfrak{A}}_{\zeta} = v}{[\![\textbf{try } R \textbf{ catch } f(t_1, \ldots, t_n)\ S]\!]^{\mathfrak{A}}_{\zeta} = v} \quad \begin{array}{l} \exists\, v_1 \neq v_2 : (loc, v_1) \in u \wedge (loc, v_2) \in u \\ where \;\; loc = f\langle [\![t_1]\!]^{\mathfrak{A}}_{\zeta}, \ldots, [\![t_n]\!]^{\mathfrak{A}}_{\zeta} \rangle \end{array}$$

Remark: The second rule for *choose* reflects the decision in [13] that an ASM does nothing when there is no choice. Obviously also other decisions could be formalized in this manner, e.g. yielding instead of the empty set an update set which contains an error report.

Remark: The rule for *forall* is formulated as finitary rule, i.e. it can be applied only for quantifying over finite sets. The set theoretic formulation in Section 2 is more general and can be formalized by an infinitary rule. It would be quite interesting to study different classes of ASMs, corresponding to different finitary or infinitary versions of the *forall* construct.

References

1. J. R. Abrial. *The B-Book. Assigning Programs to Meanings*. Cambridge University Press, 1996.
2. M. Anlauff. XASM – An extensible, component-based Abstract State Machines language. In Y. Gurevich, M. Odersky, and L. Thiele, editors, *Proc. ASM 2000*, Lecture Notes in Computer Science. Springer-Verlag, 2000. to appear.
3. C. Böhm and G. Jacopini. Flow diagrams, Turing Machines, and languages with only two formation rules. *Communications of the ACM*, 9(5):366–371, 1966.
4. E. Börger. High level system design and analysis using Abstract State Machines. In D. Hutter, W. Stephan, P. Traverso, and M. Ullmann, editors, *Current Trends in Applied Formal Methods (FM-Trends 98)*, number 1641 in Lecture Notes in Computer Science, pages 1–43. Springer-Verlag, 1999.
5. E. Börger, A. Cavarra, and E. Riccobene. An ASM semantics for UML Activity Diagrams. In T. Rust, editor, *Proc. AMAST 2000*, Lecture Notes in Computer Science. Springer-Verlag, 2000.
6. E. Börger, A. Cavarra, and E. Riccobene. A simple formal model for UML State Machines. In Y. Gurevich, M. Odersky, and L. Thiele, editors, *Proc. ASM 2000*, Lecture Notes in Computer Science. Springer-Verlag, 2000. to appear.
7. E. Börger, E. Grädel, and Y. Gurevich. *The Classical Decision Problem*. Perspectives in Mathematical Logic. Springer-Verlag, 1997.

8. E. Börger, J. Schmid, W. Schulte, and R. Stärk. *Java and the Java Virtual Machine*. Lecture Notes in Computer Science. Springer-Verlag, 2000. to appear.
9. E. Börger and W. Schulte. Modular Design for the Java Virtual Machine Architecture. In E. Börger, editor, *Architecture Design and Validation Methods*, pages 297–357. Springer-Verlag, 2000.
10. A. Brüggemann, L. Priese, D. Rödding, and R. Schätz. Modular decomposition of automata. In E. Börger, G. Hasenjäger, and D. Rödding, editors, *Logic and Machines: Decision Problems and Complexity*, number 171 in Lecture Notes in Computer Science, pages 198–236. Springer-Verlag, 1984.
11. G. D. Castillo. *ASM-SL, a Specification Language based on Gurevich's Abstract State Machines*, 1999.
12. M. Davis. *The Universal Computer: The Road from Leibniz to Turing*. W.W. Norton, New York, 2000. to appear.
13. Y. Gurevich. Evolving Algebras 1993: Lipari Guide. In E. Börger, editor, *Specification and Validation Methods*, pages 9–36. Oxford University Press, 1995.
14. Y. Gurevich. Sequential Abstract State Machines capture sequential algorithms. *ACM Transactions on Computational Logic*, 1(1), 2000.
15. Y. Gurevich and M. Spielmann. Recursive abstract state machines. *Journal of Universal Computer Science*, 3(4):233–246, 1997.
16. S. C. Kleene. *Introduction to Metamathematics*. D. van Nostrand, Princeton, New Jersey, 1952.
17. D. L. Parnas. Information distribution aspects of design methodology. In *Information Processing 71*, pages 339–344. North Holland Publishing Company, 1972.
18. J. Schmid. Executing ASM specifications with AsmGofer. Web pages at: http://-www.tydo.de/AsmGofer, 1999.
19. A. Zamulin. Object-oriented Abstract State Machines. In *Proceedings of the 28th Annual Conference of the German Society of Computer Science*. Technical Report, Magdeburg University, 1998.

Une tentative malheureuse de construire une structure éliminant rapidement les quanteurs

Bruno Poizat

Institut Girard Desargues
Mathématiques, bâtiment 101
Université Claude Bernard (Lyon-1)
43, boulevard du 11 novembre 1918
69622 Villeurbanne cedex, France
poizat@desargues.univ-lyon1.fr

Introduction

Je cherche à construire une structure M éliminant rapidement les quanteurs: à toute formule existentielle $(\exists \boldsymbol{y})\phi(\boldsymbol{x}, \boldsymbol{y})$, où ϕ est sans quanteurs, doit être associée une formule $\psi(\boldsymbol{x})$, également libre de quanteurs, qui lui soit équivalente dans M, la longueur de ψ étant polynomialement bornée en fonction de celle de ϕ; autrement dit, pour une certaine constante c, $|\psi| \leq |\phi|^c$. Tant qu'à faire, j'aimerais aussi que ψ fût calculée par un algorithme polynomial à partir de ϕ.

Pourquoi je veux faire ça? Parce que si c'était vrai de la structure à deux éléments $M = \{0, 1\}$, dans le langage réduit à l'égalité et aux deux constantes 0 et 1 (ou même à la seule relation $x = 0$!), eh bien on répondrait positivement à deux questions ouvertes en complexité, $NC^1 = P$ et $P = NP$, version uniforme ou non suivant que l'élimination serait algorithmique ou pas (que ça implique $P = NC^1$ vient de ce que j'écris les formules sous la forme usuelle, et non pas comme des circuits booléens). La même chose se produirait si la structure était finie, hormis le cas trivial d'une structure à un élément que nous excluons, ou bien si son langage fini ne comportait que des relations et pas de fonctions (voir [7]).

Donc, ce que je veux construire, c'est une structure ayant une propriété qui, si elle était possédée par la structure à deux éléments, provoquerait un cataclysme en Théorie de la Complexité! Comme vous le voyez, je m'exerce au passe-temps favori des complexionistes, qui consiste à ne démontrer des théorèmes qu'après les avoir transportés dans des contextes où ils n'ont plus aucune signification. Il est en effet très improbable qu'une structure construite spécialement pour répondre à cette question ait un quelconque enjeu algorithmique, si bien que l'intérêt du problème ne réside que dans la plus ou moins grande difficulté de sa solution.

Si on autorise un langage infini, il devient complètement trivial: prenons un modèle M de la théorie des ensembles, par exemple celui des ensembles héréditairement finis, formé des entiers naturels munis de l'appartenance d'Ackermann, dite aussi relation des bits, s'il est permis de s'exprimer ainsi ($x \in y$ s'il y a un 1 à la x° place du développement de y en base 2); la fonction qui

P. Clote and H. Schwichtenberg (Eds.): CSL 2000, LNCS 1862, pp. 61–70, 2000.

à (x, y) associe le couple $\langle x, y\rangle = \{\{x\}, \{x, y\}\}$ permet de représenter le uple $(x_1, \ldots, x_n)$ comme la "liste" $\langle x_1, \langle x_2, \langle \cdots \langle x_n, 0\rangle \cdots \rangle\rangle\rangle$; une fois identifiés à certains éléments de M les symboles servant à l'écriture des formules $\phi(\boldsymbol{x})$, toute formule $\phi(\boldsymbol{a})$ à paramètres $\boldsymbol{a}$ dans M se représente ainsi par un élément "$\phi(\boldsymbol{a})$" de M, qu'en maths modernes, où tout est ensemble, on considère comme $\phi(\boldsymbol{a})$ elle-même plutôt que comme un codage qui en respecte la taille. Nous considérons alors la structure dont l'univers est M, dans le langage comprenant (outre l'égalité) la constante 0, la fonction binaire $\langle x, y\rangle$, et une suite infinie de prédicat unaires $V_1, \ldots, V_n, V_{n+1}, \ldots$ construite par récurrence, V_{n+1} étant l'ensemble des formules $\phi(\boldsymbol{a})$ vérifiées par M qui ne font intervenir que les n premiers prédicats $V_1, \ldots, V_n$: la formule $\phi(\boldsymbol{x})$, quel que soit son rang de quantification, est équivalente dans M à V_{n+1}("$\phi(\boldsymbol{x})$"), laquelle est sans quanteurs, et pratiquement de même longueur que ϕ! Il y a aussi une manière encore plus débile d'éliminer, la "morleysation" d'une quelconque structure, qui associe à chaque formule $\phi(\boldsymbol{x})$ un nouveau symbole relationnel $R_\phi(\boldsymbol{x})$ (voir [6] p. 89).

Si on veut que le problème présente quelque difficulté, et par conséquent quelqu'intérêt, il est indispensable de se limiter à un langage fini. Il faut alors se garder des structures M trop riches en pouvoir d'auto-expression, comme l'appartenance entre ensembles, ou même l'arithmétique, qui permettent de représenter les formules par des éléments de M et de les manipuler (ce que ne permet pas la fonction de couple à elle seule; mais rien ne nous empêchait d'ajouter $\in$ à la structure ci-dessus), car un célèbre résultat de Tarski nous assure qu'elles ne peuvent éliminer les quanteurs (l'ensemble des codes d'énoncés Π_1 vrais n'est pas Σ_1; voir [6] ch. 7). D'autre part, au moins dans un premier temps, on essaye seulement d'éliminer un bloc de quanteurs existentiels, et pas un bloc de quanteurs alternant arbitrairement.

Nous conservons cet idée de prédicat de vérité, et nous cherchons à construire M, structure de langage fini comportant un prédicat unaire $V(x)$, ayant la propriété qu'à toute formule existentielle $(\exists \boldsymbol{y})\phi(\boldsymbol{x}, \boldsymbol{y})$ soit associé un terme $\tau_\phi(\boldsymbol{x})$, de longueur modérée par rapport à celle de ϕ, de sorte que M satisfasse:

$$(\forall \boldsymbol{x})(\exists \boldsymbol{y})\phi(\boldsymbol{x}, \boldsymbol{y}) \leftrightarrow V(\tau_\phi(\boldsymbol{x})).$$

La difficulté vient bien sûr de ce que la formule (libre) ϕ mentionne le prédicat V. On voit bien là tout l'artifice de la construction : il ne s'agit pas d'élaborer un algorithme pour vérifier quelque chose dans le monde réel; on se donne au contraire l'algorithme a priori, et on construit une vérité qui s'y adapte! Ça serait bien sûr beaucoup plus intéressant de montrer que certaines structures "naturelles" éliminent rapidement, ou bien lentement; mais il faut s'attendre à ce que les questions de ce genre soient plus ou moins équivalentes à des problèmes de complexité standards [5,3], ayant la réputation d'être insolubles.

Et même cette réponse artificielle, je suis incapable de la donner entièrement. Je vais seulement construire ici M satisfaisant:

$$(\forall x)(\exists \boldsymbol{y})\phi(x, \boldsymbol{y}) \leftrightarrow V(\tau_\phi(x))$$

pour les formules existentielles où il ne reste *qu'une seule* variable libre x. Cela vient de ce que j'ai besoin d'un contexte modèle théorique facilement maîtrisable,

et que je n'ai pas trouvé mieux que de me limiter à des fonctions unaires, qui ne permettent pas la formation de termes en plusieurs variables. Et c'est déjà assez compliqué comme ça!

Nous examinerons en conclusion quelques propriétés algorithmiques de la structure construite, et nous explorerons une voie qui pourrait mener à la découverte d'une autre structure plus satisfaisante. Pour l'instant, place à la construction!

1 Deux successeurs

Nous adoptons un langage L comprenant trois symboles de fonctions unaires s_0, s_1, p, et un prédicat unaire V. Les deux premières fonctions sont appelées successeurs, et la troisième prédécesseur. Fixons-en dès à présent l'interprétation.

Le modèle de base S est l'algèbre libre, dans le langage de deux fonctions unaires, engendrée par un élément r qu'on appellera sa racine; les éléments de S s'écrivent sous la forme $\theta(r)$, où θ est un terme en s_0, s_1, $\theta = s_{\epsilon_1} \circ s_{\epsilon_2} \circ \cdots \circ s_{\epsilon_m}$. La longueur m du terme θ sera aussi appelée niveau de l'élément $\theta(r)$, la racine r étant de niveau nul. Si $\theta(r) = \theta'(r)$, c'est que les deux termes θ et θ' sont identiques. S a la forme d'un arbre binaire infini, composé de sa racine r, de ses deux successeurs $s_0(r)$ et $s_1(r)$, puis des quatre successeurs d'iceux $s_0(s_0(r))$, $s_0(s_1(r))$, $s_1(s_0(r))$ et $s_1(s_1(r))$, et ainsi de suite.

Le prédécesseur est l'inverse des successeurs, c'est-à-dire que $p(s_0(x)) = x$, $p(s_1(x)) = x$, $p(r) = r$.

Nous appelons bloc B une copie de S muni d'un quelconque prédicat unaire V_B. Nous considérons la classe C des structures M, de langage L, formées de la juxtaposition de blocs qui se répètent chacun une infinité de fois: si B figure dans M, il s'y trouve aussi une infinité de blocs isomorphes à B.

Les structures de C éliminent les quanteurs: c'est très facile à voir par des méthodes modèle-théoriques impliquant la compacité. Mais nous, étant dans l'obligation de contrôler la taille des formules éliminantes, n'avons d'autre choix que de procéder péniblement à cette élimination.

Si n est un entier, et x un élément d'un bloc, nous appelons triangle de hauteur n et de racine x la formule caractérisant à l'isomorphie près le morceau du bloc composé des éléments qui sont situés au plus n niveaux au-dessus de x; c'est la conjonction des formules satisfaites par x, de la forme $p(x) = x$ ou $p(x) \neq x$, $V(x)$ ou $\neg V(x)$, et $V(y)$ ou $\neg V(y)$ pour chaque $y = s_{\epsilon_1} \circ s_{\epsilon_2} \circ \cdots \circ s_{\epsilon_m}(x)$, $m \leq n$. Le n-voisinage de x, c'est le triangle de racine $p^n(x)$ et de hauteur $2n$.

Lemma 1 (Lemme d'élimination). *Considérons M dans C, une formule existentielle $(\exists \boldsymbol{y})\phi(\boldsymbol{x}, \boldsymbol{y})$, et l'entier n valant quatre fois le nombre total d'occurrences dans ϕ des symboles s_0, s_1 et p. Alors, pour déterminer si un élément x de M satisfait cette formule, il suffit de connaître le n-voisinage de x, et de savoir quels sont les n-triangles réalisés dans M.*

Nous appelerons *portée* de la formule ϕ ce nombre n figurant dans l'énoncé du lemme.

Demonstration 1. Nous commençons par mettre ϕ sous forme disjonctive, ce qui fait exploser sa taille, mais pas celles des constituants: ϕ s'écrit comme une disjonction exponentielle de conjonctions où figurent exactement une fois, sous forme positive ou négative, chacune des formules atomiques de ϕ: dans chaque conjonction figure le même nombre ω de symboles de fonction que dans ϕ. Comme le quanteur existentiel saute au-dessus de la disjonction, il suffit de traiter chacune de ces conjonctions $\phi_i(x, \boldsymbol{y})$.

Les constituants de $\phi_i(x, \boldsymbol{y})$ sont de la forme $V(\theta(u))$, $\neg V(\theta(v))$, $\theta'(u) = \theta''(v)$, $\theta'(u) \neq \theta''(v)$, où u et v désignent des variables, distinctes ou confondues, prises dans le uple $x^\frown \boldsymbol{y}$, et où $\theta, \theta', \theta''$ désignent des termes du langage L, en s_0, s_1 et p.

Nous dirons qu'une variable v prise dans $\boldsymbol{y}$ est à distance 1 de x si il y a dans ϕ_i une équation $\theta'(v) = \theta''(x)$; qu'elle est à distance 2 de x si elle n'en est pas à distance 1, mais figure dans une équation $\theta'(v) = \theta''(u)$, où u est à distance 1 de x; et plus généralement, on dit qu'une variable v est voisine de x si elle est reliée à cette dernière par une chaîne d'équations figurant dans ϕ_i, sa distance à x étant la longueur minimale d'une telle chaîne.

Pour chaque voisine de x, nous faisons le choix d'une équation de ϕ_i qui la relie à x, ou bien à une variable strictement plus proche de x. Nous notons E la conjonction des équations choisies.

Pour éliminer, on procède ainsi: on écrit chaque équation $\theta'(v) = \theta''(u)$ de E, dont v est la variable la plus lointaine, sous la forme $v = \theta(u)$; cela peut obliger à distinguer un nombre exponentiel de cas, s'exprimant par des conditions sur u, ce qui a pour influence d'enfler encore la disjonction, et d'ajouter quelques conditions à chacune des conjonctions. Par exemple, si l'équation est $p(v) = u$, il faut distinguer les trois cas $u = p(u) = v$, $v = s_0(u)$, $v = s_1(u)$. Ce qui importe, c'est que la taille de E n'est pas affectée au cours du procès, et que dans les conditions ajoutées les termes n'ont pas une longueur supérieure à ω.

Quand on a fini, le système E permet d'exprimer chaque voisine v de x comme un terme en une variable plus proche; comme le nombre total de symboles de fonction de E est majoré par ω, v se met par composition sous la forme $v = \eta(x)$ où le terme η est de longueur au plus ω. On substitue alors partout les variables voisines de x par leur valeur, ce qui ne produit que des termes de longueur majorée par 2ω.

Quand on a ainsi éliminé les voisines de x, chaque système $\psi(x, \boldsymbol{y})$ obtenu est formé de trois parties:

- une conjonction $\psi_0(x)$ de conditions portant sur des termes en x de longueur inférieure à 2ω;
- une conjonction $\psi_1(x, \boldsymbol{y})$ d'inéquations $\theta'(x) \neq \theta''(y)$;
- une conjonction $\psi_2(\boldsymbol{y})$ de conditions portant sur des variables qui n'ont pas été touchées par l'élimination, et qui étaient toutes présentes à l'origine: dans ψ_2, il n'y a pas plus de ω occurrences de fonctions.

De plus, l'hypothèse de répétition des blocs nous permet de jeter ψ_1; en effet, $(\exists \boldsymbol{y})\psi(x, \boldsymbol{y})$ équivaut à $\psi_0(x) \wedge (\exists \boldsymbol{y})\psi_2(\boldsymbol{y})$, puisque s'il existe des $\boldsymbol{y}$ dans M satisfaisant ψ_2, il en existe dont aucun élément ne soit dans le bloc de x.

On choisit alors une variable dans $\boldsymbol{y}$ par rapport à laquelle on applique le même procédé. A la fin on obtient une décomposition de la formule originelle en une monstrueuse disjonction de formules du type $\phi_0(x) \wedge (\exists y_1)\phi_1(y_1) \wedge \cdots \wedge (\exists y_m)\phi_m(y_m)$; comme tous les termes qui y figurent ont leur longueur majorée par 2ω, leur satisfaction ne dépend que de ce qui est indiqué dans l'énoncé du lemme. Fin

Remarques

1. Ce n'est pas ici qu'il est essentiel que x soit un élément et non un uple; l'élimination fonctionne aussi bien pour un uple.
2. Ehud Hrushovski m'a suggéré d'employer des fonctions-successeurs dans ce contexte, et Farzad Didehvar a contribué à l'élimination.

2 Le prédicat de vérité

Nous choisissons maintenant le terme τ_ϕ. Les habitants de la ville de Sour, au sud du Liban, ont inventé il y a quelques 45 siècles un système très pratique de représentation graphique de la parole comportant seulement 22 signes, qu'ils ont appelé alphabet, et qu'un citoyen américain a réduit bien plus tard à deux symboles. Cela nous permet d'écrire une formule ϕ comme un mot binaire $(\epsilon_1, \ldots, \epsilon_m)$, $\epsilon_i = 0$ ou 1. Nous demandons que la longueur de ce mot vaille au moins 72, et soit aussi supérieure ou égale à $24n$, où $n = 4\omega$ est l'entier qui a été appelé portée de ϕ, et qui figure dans le lemme d'élimination. Vu la présence des parenthèses, la traduction en Morse et la corrélation entre la portée et le nombre de symboles de la formule, ça ne demande pas beaucoup de bourrage!

Par ailleurs, pour nous faciliter la vie, nous demandons aussi qu'il y ait un signal de début de formule: toute formule commence par 00, et on n'y trouve ensuite plus jamais deux 0 consécutifs. J'aurais pu simplifier légèrement la présentation en introduisant un troisième successeur, spécialisé dans le rôle de caractère de début de formule: comme ce n'était pas décisif, j'en ai fait l'économie.

Cela étant convenu, à $\phi = (\epsilon_1, \ldots, \epsilon_m)$ nous associons tout simplement le terme $\tau_\phi = s_{\epsilon_m} \circ \cdots \circ s_{\epsilon_1}$.

Nous dirons qu'un élément y d'une structure dans C est contraint s'il est de la forme $y = \tau_\phi(x)$, où ϕ est une formule, et qu'il est libre sinon. Si y est contraint, il ne l'est que d'une seule manière: si $\tau_\phi(x) = \tau_\psi(x')$, alors $\phi = \psi$, et $x = x'$, puisque, pour trouver la formule qui contraint éventuellement y, on en remonte les prédécesseurs jusqu'à ce qu'on trouve deux 0 consécutifs. Ce n'est d'ailleurs pas là que le signal de début s'impose, puisque les parenthèses, garantes de la non ambiguïté de lecture, interdisent à une formule d'être segment final d'une autre. Son utilité véritable, c'est de forcer les formules qui passent par y, à l'exception de celles qui concernent son prédécesseur (dans le cas où $y = s_0(p(y))$), de concerner toutes un même élément.

On voit la nécessité d'au moins deux successeurs, non seulement pour avoir une longueur raisonnable pour le terme τ_ϕ, mais aussi pour la non ambiguïté de

la lecture des formules: il est clair que des confusions du genre $\tau_\phi(x) = \tau_\psi(x')$ handicaperaient lourdement la construction du prédicat de vérité.

Ce prédicat V est construit niveau par niveau à partir du bas. Pour ce qui est des points libres du niveau en construction, on met toutes les possibilités d'appartenance et de non appartenance à V, ce qui conduit à démultiplier ce qui a déjà été construit en plusieurs blocs; c'est ainsi qu'au début, jusqu'au niveau 71, il n'y a que des points libres, si bien qu'on trouve au bas des divers blocs des copies de tous les triangles de hauteur 71 possibles.

Par contre, quand il s'agit de placer dans V ou dans $\neg V$ un point de la forme $\tau_\phi(x)$, où x est au moins $24n$ niveaux en dessous, nous devons faire un pari sur l'avenir, c'est-à-dire anticiper de la satisfaction, ou de la non-satisfaction, de la formule $(\exists \boldsymbol{y})\phi(x, \boldsymbol{y})$ par l'élément x dans la structure en cours de construction. Nous connaissons le n-voisinage de x, obtenu aux niveaux antérieurs, mais l'information qui nous manque, c'est de savoir quel sont les n-triangles qu'on trouvera dans la structure quand la construction en sera achevée. *Nous faisons comme si nous les avions tous déjà obtenus*, plus précisément, nous faisons l'hypothèse qu'il n'y aura pas d'autres n-triangles présents dans cette structure que ceux qui y ont été obtenus en dessous du niveau $24n$ (pour $n \geq 1$; les formules de portée nulle ne posent pas de problèmes, car les quatre triangles possibles de hauteur 0 sont tous obtenus aux niveaux 0 ou 1). D'après le lemme d'élimination, cette hypothèse nous permet de définir une stratégie pour déterminer la satisfaction de $(\exists \boldsymbol{y})\phi(x, \boldsymbol{y})$: si c'est oui, on met $\tau_\phi(x)$ dans V, et sinon en dehors. Quand on a fini, on démultiplie tous les blocs de la structure obtenue.

3 L'art de faire des hypothèses

C'est bien beau de faire des hypothèses, à condition qu'elles se vérifient. Pour que la vérité devinée corresponde à la vérité vraie, nous devons montrer que le modèle M obtenu satisfait à l'hypothèse qui a été faite lors de sa construction, à savoir que tout triangle de hauteur $n \geq 1$ qui s'y trouve y a une copie isomorphe située toute entière en-dessous du niveau $24n$. C'est une conséquence du lemme suivant, puisque $6n + n < 24n < 24(n + 1)$.

Lemma 2 (Lemme de descente). *Soit x dans M de niveau strictement supérieur à $6n$; alors ils existe x' dans M, de niveau au plus $6n$, tel que:*

1. *le triangle de racine x', et montant jusqu'au niveau $24(n + 1) - 1$, est isomorphe à celui de même hauteur et de racine x*
2. *le chemin obtenu en prenant les prédécesseurs de x' passe par $s_1(r)$, et non pas par $s_0(r)$, où r est la racine du bloc de x'; il est isomorphe au chemin de longueur correspondante partant de x (c'est-à-dire que si $s_1(r) = p^s(x')$, c'est le même terme qui exprime x' en fonction de $p^s(x')$ et x en fonction de $p^s(x)$).*

Demonstration 2. Par induction sur n. Pour $n \leq 2$, on peut remplir comme on veut le triangle au-dessus de $x' = s_1(r)$ tant qu'on reste en dessous du niveau 72, puisque tous les points y sont libres.

Soit donc $n \geq 3$, et soit m la partie entière de $n/3$: $3m \leq n < 3(m+1)$. Comme $1 \leq m < n$, l'hypothèse d'induction s'applique à m.

Soit donc x dans M, de niveau supérieur à $6n$. Nous voulons choisir x', de niveau au plus $6n$, tel que le triangle au-dessus de x' se soit rempli lors de la construction de M de la même façon que le triangle au-dessus de x, cela jusqu'à ce qu'on atteigne le niveau $24(n+1)-1$. Les points libres au-dessus de x' ne nous gêneront pas, puisque nous avons eu la possibilité de les remplir comme on l'a voulu. Quant au points contraints, comme nous restons en dessous du niveau $24(n+1)$, ils correspondent à des formules ϕ de portée au plus n: par la construction même de M, si deux points u et v de M ont des n-voisinages isomorphes, et si le premier satisfait $V(\tau_\phi(u))$, l'autre aussi, et réciproquement.

Nous devons tenir compte des points contraints au-dessus de x; certains concernent le prédécesseur de x, ou bien x lui-même, ou encore des éléments au-dessus de x. Si on ne rencontre que ceux-là, il suffira que le n-voisinage de $p(x')$ soit choisi isomorphe à celui de $p(x)$, celui de x' isomorphe à celui de x, et celui d'un point v', situé moins de n niveaux au-dessus de x', isomorphe à celui du point v correspondant situé au-dessus de x; en effet, comme les points situés plus haut dans les triangles ont leur n-voisinage entièrement situé au-dessus de x et x', le triangle au-dessus de x' se remplira ensuite de manière isomorphe à celui au-dessus de x.

Mais il est aussi possible que x soit sur le trajet de formules beaucoup plus longues, correspondant toutes à un unique élément u, le premier que l'on rencontre après un double 0 quand on chemine le long des prédécesseurs de x. Dans ce cas, nous dirons que x a une queue – le chemin qui joint x à u – et il faudra aussi nous soucier de reproduire le n-voisinage de u.

Nous distinguons deux cas :

1. x n'a pas de queue, ou bien a une queue de longueur strictement supérieure à $n+6m$. Dans ce cas on choisit y' de niveau au plus $3m$ associé par la m-ième étape du lemme à $y = p^{n+1}(x)$. Soit x' l'élément correspondant à x. Comme $12m+n+1+2n < 24(m+1)$, $p(x)$ et $p(x')$ ont des n-voisinages isomorphes, ainsi que x et x', et aussi les points v et v' situés moins de n niveaux au-dessus; par ailleurs x' n'a pas de queue (on la lui a coupée), de sorte que les points éventuellement contraints par la longue queue de x sont maintenant libres, et qu'on peut les mettre dans V ou dans $\neg V$ ad libitum, si bien qu'il y a une façon de faire qui donne le même triangle au-dessus de x' qu'au-dessus de x, tant qu'on reste en-dessous du niveau $24(n+1)$.
2. x a une queue de longueur inférieure ou égale à $n+6m$. Dans ce cas-là, nous devons reproduire les n-voisinages de x, de $p(x)$, des points v, mais aussi du point u qui est au bout de la queue, et pour cela il nous suffit de reproduire grâce à l'hypothèse de récurrence le triangle de racine $p^n(u)$ et de hauteur $n+6m+n+2n$ en-dessous du niveau $24(m+1)$: ça marche parce que $12m+4n < 24(m+1)$.

Nous avons recopié x en x' à un niveau majoré par $6m+n+1$ dans le premier cas, $12m+2n$ dans le second. Il ne reste plus qu'à observer que $2n+12m \leq 6n$. Fin

Remarque. Quand le lapin blanc est sorti du chapeau, on aime bien connaître le truc. Cette démonstration dépend de trois paramètres A, B et C: on demande que la longueur des formules soit au moins An; on cherche à reproduire les racines des triangles de hauteur n en dessous du niveau Bn; pour le pas de récurrence, on divise n par C, ce qui demande AC comme longueur minimum de formule. On pose $m = \lfloor n/C \rfloor$, et on coupe les queues à $n + Bm$; le premier cas demande que $Bm + n + 1 + 2n < A(m+1)$, le second que $2Bm + 4n < A(m+1)$, et la conclusion que $2Bm + 2n \leq Bn$. Cela donne $2B + 2C \leq A$, $2B/C + 2 \leq B$, soit encore, pour $C \geq 2$, $B \geq 2C/C - 2$, $A \geq 4C(C-1)/C - 2$; le minimum pour A est atteint en $C = 2 + \sqrt{2}$, et vaut $12 + 8\sqrt{2}$. J'ai choisi les paramètres $C = 3$, $B = 6$, $A = 24$, valeur très proche du minimum; d'autres choix raisonnables sont $C = 4$, $B = 4$, $A = 24$, ou $C = 6$, $B = 3$, $A = 30$. Les grandes valeurs de C – et de A ! – font tendre au contraire B vers sa borne inférieure.

Conclusion

Si on ajoute au langage de M une constante r nommant une racine ($p(r) = r$), on obtient une structure qui a un algorithme de décision rapide pour ses énoncés existentiels, puisque $(\exists \boldsymbol{y})\phi(r, \boldsymbol{y})$ est vrai si et seulement si $V(\tau_\phi(r))$ l'est: pour voir si l'énoncé est vrai, on calcule $\tau_\phi(r)$ et on teste s'il satisfait V! C'est peut-être pas bien convaincant comme exemple, puiqu'il s'agit d'un "algorithme" au sens de la structure M, où, suivant les conventions de [1] ou [7], ou plus généralement de la Complexité Algébrique, on considère par définition comme algorithmique l'évaluation d'une formule (et même d'un circuit) libre de quanteurs. Je ne crois pas beaucoup améliorer mon cas en faisant observer que le problème analogue pour la structure $\{0, 1\}$ est NP-complet.

Cependant, on pourra poser $0 = s_0(r)$ et $1 = s_1(r)$ pour respecter les conventions de [7]; comme un énoncé n'est rien d'autre qu'un objet syntaxique, la propriété décrite au paragraphe ci-dessus s'interprète ainsi: tout problème booléen qui est NP au sens de M est P au sens de M! Il est très facile de construire des structures ou n'importe quel problème booléen est P grâce à l'intervention d'un paramètre approprié (voir les "dictionnaires" de [4], mais dans le cas présent, il n'est nul besoin d'ajouter des paramètres pour transformer un algorithme NP en un algorithme P ayant même effet sur des données booléennes: je ne connais pas de méthode de construction plus simple pour obtenir cette propriété.

On observe aussi que la formule $V(\tau_\phi(x))$ est horriblement séquentielle, alors que j'ai annoncé dans l'introduction un contrepoint à $NC^1 = P$! Cela vient de ce que, puisqu'il y a des fonctions, il n'est plus possible de paralléliser les formules, si bien que la connection entre "formules écrites sous forme traditionnelle" et "circuits de profondeur logarithmique" est perdue!

Je conjecture que notre modèle M n'élimine plus rapidement dès qu'il reste deux variables libres: la candidate à une élimination difficile, c'est la formule

existentielle en les deux variables libres x et x' qui exprime le fait qu'il existe $\epsilon_1, \ldots, \epsilon_n$ tels que soient vérifiées $V(s_{\epsilon_1}, \ldots, s_{\epsilon_n}(x))$ et $\neg V(s_{\epsilon_1}, \ldots, s_{\epsilon_n}(x'))$; elle s'écrit:

$$(\exists y_1)\cdots(\exists y_n)(\exists z_1)\cdots(\exists z_n)$$
$$((y_1 = s_0(x) \wedge z_1 = s_0(x')) \vee (y_1 = s_1(x) \wedge z_1 = s_1(x'))) \wedge \cdots \wedge$$
$$((y_{i+1} = s_0(y_i) \wedge z_{i+1} = s_0(z_i)) \vee (y_{i+1} = s_1(y_i) \wedge z_{i+1} = s_1(z_i))) \wedge \cdots \wedge$$
$$V(y_n) \wedge \neg V(z_n).$$

Il est probable qu'on ne puisse décider de la véracité de cette formule sans faire dans les pires cas beaucoup de tests, c'est-à-dire, selon les mots de [3], qu'il n'y ait pas d'arbres de décision de profondeur polynomiale pour les problèmes NP au sens de M. Je pense donc que M satisfait $P \neq NP$, et même $P \neq NBP$. Pour en avoir le coeur net, il faudrait examiner le problème suivant: soit B_0 un bloc de M dont tous les points libres sont hors de V; si je perturbe sa construction en mettant un certain point libre dans V, quels sont les points contraints qui vont être touchés? Combien y en aura-t'il à un niveau donné?

Comme nous l'avons dit, la faiblesse de la méthode vient de ce qu'elle n'utilise que des fonctions unaires. Pour éliminer rapidement les quanteurs sans restriction sur $\boldsymbol{x}$, la tentation est de coller un prédicat de vérité sur la fonction-couple de la Théorie des Ensembles, qui n'est autre qu'une algèbre libre, à une infinité de générateurs, dans le langage d'une fonction binaire (c'est un exemple d'algèbre de Malcev). C'est cette méthode qui a donné facilement dans [7] p. 188 un exemple de structure où $P = NBP$; mais ça risque d'être beaucoup plus dur quand on est en présence de quanteurs véritables, car si la théorie de l'algèbre de Malcev elle-même est facilement contrôlable [2], elle devient sauvage quand on lui ajoute un prédicat unaire arbitraire, puisque l'arité unaire n'est pas plus simple qu'une autre en présence d'une bijection entre l'univers et son carré cartésien!

Ce dont nous avons profité, c'est que la théorie des modèles pour deux successeurs est non seulement absolument triviale, mais qu'elle le reste si on introduit des prédicats unaires à peu près arbitraires. C'est ça qui nous a permis de prévoir le comportement final de l'objet que nous étions en train de construire. Dans cette nouvelle situation infiniment plus complexe, il nous faudrait une classe de prédicats qui restent sous contrôle quand on les plaque sur la fonction-couple, tout en laissant assez de liberté de choix pour pouvoir représenter la vérité. Ça n'a pas l'air si simple, et je n'ai pas de candidat; d'un autre côté, je ne vois aucune raison métaphysique qui interdise l'existence d'une telle structure, et je suis étonné de n'avoir jamais rencontré dans mes lectures de tentative d'en fabriquer une: vu la célébrité des résultats de Gödel et de Tarski, affirmant l'impossibilité de définir les énoncés prouvables ou les énoncés vrais, il est surprenant de constater que personne n'ait voulu, par la construction de contre-exemples, en délimiter précisément le cadre de validité.

L'idéal, ça serait d'exhiber $\tau_\phi(\boldsymbol{x})$ de longueur logarithmique, et de construire V tel que $(\exists \boldsymbol{y})\phi(\boldsymbol{x}, \boldsymbol{y})$ équivaille à $V(\tau_\phi(\boldsymbol{x}))$.

On pourrait aussi chercher à éliminer en suivant de tout autres principes, par exemple en déterminant un $\boldsymbol{y}$, s'il en existe, parmi ceux qui satisfont $\phi(\boldsymbol{x}, \boldsymbol{y})$ (à ce propos, la question 1 au bas de la p. 181 de [7] est complètement stupide, puisque

dans un corps fini toute fonction est polynôme; comme le montre le lemme d'élimination, on a également des fonctions de Skolem décrites par des termes dans le langage de un ou plusieurs successeurs, augmenté du sélecteur et de la fonction caractéristique de l'égalité; mais le problème est celui de la complexité du uple de termes associé à la formule ϕ), ou bien en décrivant une bonne raison expliquant si oui ou non il y a un $\boldsymbol{y}$ qui satisfait $\phi(\boldsymbol{x}, \boldsymbol{y})$. Ce serait quelque chose qui ressemblerait d'avantage à l'idée qu'on se fait d'un algorithme d'élimination que la simple consultation d'un oracle miraculeux. Il est aussi possible que ce que j'ai fait ici soit naïvement compliqué, et que certaines structures très simples et bien connues éliminent de toute évidence et à une vitesse foudroyante; mais j'en doute.

References

1. Blum, L., Cucker, F., Shub, M., Smale, S.: Complexity and real computations. *Springer-Verlag* (1998).
2. Bouscaren, E., Poizat, B.: Des belles paires aux beaux uples. *Journal of Symbolic Logic* **53** (1988) 434–442.
3. Fournier, H., Koiran, P.: Lower bounds are not easier for the Reals. Prépublications LIP-ENS Lyon, Number 1999-21 (1999).
4. Goode, J.B.: Accessible telephone directories. *Journal of Symbolic Logic*, **59** (1994) 92–105.
5. Koiran, P.: Transfer theorems via sign conditions. Prépublications LIP-ENS Lyon, Number 2000-13 (2000).
6. Poizat, B.: *Cours de Théorie des Modèles*. Nur al-Mantiq wal-Maŕifah. Version anglaise par Moses Klein, *A Course in Model Theory*, *Springer-Verlag* (2000).
7. Poizat, B.: *Les petits cailloux*. Nur al-Mantiq wal-Márifah. Aléas Editeur, 15 quai Lassagne, 69001 Lyon, France.

Translating Theory into Practice — Abstract State Machines within Microsoft

Wolfram Schulte

Microsoft Research
One Microsoft Way, Redmond, WA, 98052-6399, USA
Phone: +1 425 703 4641, schulte@microsoft.com
http://research.microsoft.com/users/schulte/

As researchers in rigorous methods we are interested in the challenge of building and maintaining software systems on a massive scale. Our methods must work for real-world systems with hundreds or thousands of components that interact in complex ways.

To this end, Microsoft Research has recently developed a new specification language called ASML. It is based on Abstract State Machines and tailored to inter-operate with Microsoft runtime environments and languages.

Specifications written in ASML are executable. They expose the same interfaces as the actual implementation, and are packaged as programs or libraries. Accordingly, ASML can be integrated into Microsoft runtime environments and inter-operate with other other languages. Developers can use them from within Microsoft's Visual Studio integrated development environment, and specification writers can include ASML in other project-related documents, such as those stored as Microsoft Word files or HTML. The compiler extracts the ASML source from the text document to execute the model.

In this way ASML can be used to:

- Explore user scenarios interactively from within a test harness.
- Explore the proposed functionality in a live environment.
- Check the implementation against the specification.

In this abstract, I will only describe the last of these, namely, how to use ASML as part of the development process to check that specifications and their corresponding implementations agree.

When an implementation becomes available, it can be run in parallel with its executable specification. This involves a new kind of runtime check: we test the implementation's externally visible behavior over time against the behavior embodied in the ASML specification. This kind of runtime checking has the advantage over other approaches in being able to detect errors without instrumenting the components being tested. Even with no access to component internals, we can monitor the sequence of external component interactions, the arguments passed and the values returned. If any of the assumptions of the design are inconsistent with the observed behavior, a runtime assertion failure will occur, and the developer will receive contextual information about the run. Similarly, if the specification is in error, it will cause false runtime assertion failures. This provides a way of ensuring that specifications are always current.

P. Clote and H. Schwichtenberg (Eds.): CSL 2000, LNCS 1862, p. 71, 2000.

Choiceless Polynomial Time Logic: Inability to Express

Saharon Shelah

Institute of Mathematics
The Hebrew University
Jerusalem, Israel

Rutgers University
Mathematics Department
New Brunswick, NJ USA

Dedicated to my friend Yuri Gurevich

Abstract. We prove for the logic $\widetilde{C}PT$ime (the logic from the title) a sufficient condition for two models to be equivalent for any set of sentences which is "small" (certainly any finite set), parallel to the Ehrenfeucht Fraïssé games. This enables us to show that sentences cannot express some properties in the logic $\widetilde{C}PT$ime and prove 0-1 laws for it.

Key words and phrases. Finite model theory; Computer Science; Polynomial time logic; choiceless; games.

Partially supported by the United States-Israel Binational Science Foundation. Publication 634.

I would like to thank Alice Leonhardt for the beautiful typing.

P. Clote and H. Schwichtenberg (Eds.): CSL 2000, LNCS 1862, pp. 72-125, 2000.

Annotated Content

§0 Introduction

We deal here with choiceless polynomial time logic, introduced under the name $\widetilde{CPT}$ime in Blass Gurevich Shelah [BGSh 533]; actually we deal with several versions. Knowledge of [BGSh 533], which is phrased with ASM (abstract state machine), is not required except when we explain in §4 how the definitions here and there fit. See there more on background, in particular, on ASM-s and logic capturing P Time. The aim of this logic is to capture statements on a (finite) model M computable in polynomial time and space without arbitrary choices. So we are not allowed to choose a linear order on M, but if P is a unary predicate from the vocabulary τ_M of M and P^M has $\leq \log_2(\|M\|)$ elements then we are allowed to "create" the family of all subsets of P^M, and if e.g. $(|P^M|)! \leq \|M\|$ we can create the family permutations of P^M. Note that a statement of the form $\widetilde{CPT}$ime captures what can be computed in polynomial time without arbitrary choices" is a thesis not a theorem. For a given model M, we consider the elements of M as urelements, and build inductively $N_t = N_t[M]$, with $N_0 = M, N_{t+1} \subseteq N_t[M] \cup \mathscr{P}(N_t[M])$ but the definition is uniform and the size of $N_t[M]$ should not be too large, with the major case being: it has a polynomial bound.

So we should have a specific guide Υ telling us how to create N_{t+1} from N_t, hence we should actually write $N_t[M, \Upsilon]$. In the simplest version (we called it pure) essentially $\Upsilon = \{\psi_\ell(x, \bar{y}) : \ell < m_0\}$ and N_t is a transitive finite set with M the set of urelements, $N_0 = M, N_{t+1} = N_t \cup \{\{a : N_t \models \psi_\ell(a, \bar{b})\} : \bar{b} \in {}^{\ell g(\bar{y})}(N_t)$ and $\ell < m_0\}$; where each ψ_ℓ is a first order formula in the vocabulary of M plus the membership relation $\in$, (i.e. $\tau_M \cup \{\in\},$) and N_t has the relations of M and the relation $\in \restriction N_t$. We stop when $N_t = N_{t+1}$, in the "nice" cases after polynomial time and space, and then can ask "$N \models \chi$"? getting yes or no.

We consider several versions of the definition of the logic; this should help us to see "what is the true logic capturing polynomially computable statements without making arbitrary choices".

Our aim here is to deal with finite models and processes, but we dedicate a separate place at the end to some remarks on infinitary ones and set theory. We also comment in this section on classical model theoretic roots; both, of course, can be ignored.

For a logic $\mathscr{L}$ it is important to develop methods to analyze what it can say. Usual applications of such methods are to prove that:

- (a) no sentence in $\mathscr{L}$ expresses some property (which another given logic can express so we can prove that they are really different)
- (b) certain pairs of models are equivalent
- (c) zero-one laws.

For first order logic we use mostly elimination of quantifiers, E.F. (Ehrenfeucht-Fraïssé) games (see [CK] or [Ho93] or Ebbinghaus and Flum [EbFl95]) and others; we are most interested in relatives of E.F. games. In finite model theory people have worked on the logic $L_{\omega,k}$ (usually denoted by $L^k_{\omega,\omega}$) of first order

formulas in which every subformula has $\leq k$ free variables, a relative of the $L_{\lambda,\kappa}$ logics (see e.g. [Di]). We can use even such formulas in $L_{\infty,\omega}$ but for equivalence of finite models this does not matter. For $L_{\infty,\kappa}$ the relatives of E.F. games are called pebble games (see [EbFl95]).

The E.F. criterion (see Karp [Ka]) for the $L_{\infty,\omega}$-equivalence of two models M_1, M_2 of the same vocabulary τ is the existence of a non-empty family $\mathscr{F}$ of partial one-to-one functions from M_1 to M_2 such that for every $f \in \mathscr{F}, i \in \{1,2\}$ and $a \in M_i$ there is $g \in \mathscr{F}$ extending f such that $i = 1 \Rightarrow a \in \text{Dom}(g)$ and $i = 2 \Rightarrow a \in \text{Rang}(g)$ (a particular case is $M = N$; there, of course, M, N are equivalent but the question is when are $(M, a), (M, b)$ equivalent). Note that for finite models and even for countable models, this criterion implies isomorphism. But if we restrict ourselves to first order sentences of quantifier depth $< k$ we can replace $\mathscr{F}$ by $\langle \mathscr{F}_\ell : \ell < k \rangle$ and above we say that for every $f \in \mathscr{F}_{\ell+1}, j \in \{1,2\}, a \in M_j$ there is $g \in \mathscr{F}_\ell$ as there (this is the original E.F. game). For $L_{\infty,k}$ this does not work but without loss of generality, $f \in \mathscr{F}$ & $A \subseteq \text{Dom}(f) \Rightarrow f \restriction A \in \mathscr{F}$, and above we restrict ourselves to $f \in \mathscr{F}$ with $|\text{Dom}(f)| < k$. Now probably the simplest models are those with equality only: so every permutation of the (universe of the) model is an automorphism. So using this group it is proved in [BGSh 533] that $\widetilde{C}PT$ime cannot say much on such models, thus showing that the $\widetilde{C}PT$ime does not capture P Time logic, in fact odd/even is not captured.

But in our case suppose we are given $\mathscr{F}$, a family of partial isomorphisms from M_1 to M_2, we have to create such a family for $N_t[M_1, \Upsilon], N_t[M_2, \Upsilon]$.

We answer here some questions of [BGSh 533]: get a 0-1 law, show that $\widetilde{C}PT$ime + counting does not capture P Time.

Note that if P^M is small enough then we can have e.g. $\text{Per}(P^M)$, the group of permutations of P^M, as a member of $N_t = N_t[M, \Upsilon]$ for t large enough: just in N_{3t} we may have the set of partial permutations of P^M of cardinality $< t$.

We thank Yuri Gurevich and Andreas Blass for helpful comments.

Notation:

1) Natural numbers are denoted by $i, j, k, \ell, m, n, r, s, t$. We identify a natural number t with $\{s : s < t\}$ so $0 = \emptyset$ and we may allow $t = \infty =$ the set of natural numbers (ω for set theorists). Let $[m] = \{1, \ldots, m\}$.

2) Let τ denote a vocabulary, i.e. a set of predicates P (each predicate with a given arity $\mathbf{n}(P)$); we may also have function symbols, but then it is natural to interpret them as partial functions, so better to treat them as relations and avoid function symbols. We may attach to each predicate $P \in \tau$ a group $\mathbf{g}_P$ permutations of $\{0, \ldots, \mathbf{n}(P) - 1\}$ telling under what permutations of its argument places the predicate P is supposed to be preserved; if not specified $\mathbf{g}_P$ is a trivial group.

3) Let P, Q, R denote predicate symbols.

4) Formulas are denoted by $\varphi, \psi, \theta, \chi$; usually θ, χ are sentences.

5) Let $\mathscr{L}$ denote a logic and $\mathscr{L}(\tau)$ denote the resulting language, the set of $\mathscr{L}$-formulas in the vocabulary τ.

6) Let M, N denote models and let $\tau_M = \tau(M)$ denote the vocabulary of M

and let P^M denote the interpretation of the predicate $P \in \tau_M$. So P^M is a relation on M with arity $\mathbf{n}(P)$ and if $\mathbf{g}^P = \mathbf{g}^\tau_P$ is not trivial and $\sigma \in \mathbf{g}_P, \langle a_\ell : \ell < \mathbf{n}(P)\rangle \in {}^{\mathbf{n}(P)}M$, then $\langle a_\ell : \ell < \mathbf{n}(P)\rangle \in P^M \Leftrightarrow \langle a_{\sigma(\ell)} : \ell < \mathbf{n}(P)\rangle \in P^M$. Models are finite if not said otherwise.
7) Let $|M|$ be the universe (= set of elements) of M and $\|M\|$ the cardinality of $|M|$, but abusing notation we may say "of M".
8) Let $\mathbf{t}, \mathbf{s}$ denote functions from the set of natural numbers to the set of natural numbers ≥ 1 and let $\mathbf{T}$ denote a family of such functions. We may write $\mathbf{t}(M)$ instead of $\mathbf{t}(\|M\|)$.
9) For a function f and a set A let $f"(A) = \{f(x) : x \in A \text{ and } x \in \text{ Dom}(f)\}$. [Why do we not write $f(A)$? Because f may act both on A and on its elements; this occurs for functions like $G(f)$, defined and investigated is §2.]

0.1 Discussion: Would we really like to allow $\mathbf{t}(M)$ to depend not just on $\|M\|$? For the definition of the logic we have no problems, also for the criteria of equivalence of M_1, M_2 (except saying $\|M_1\| = \|M_1\|$ & $\mathbf{t}_1 = \mathbf{t}_2 \Rightarrow \mathbf{t}_1(M_1) = \mathbf{t}_2(M_2)$). But this is crucial in proving a weak version of the 0-1 law, say for random graphs, to overcome this we need to "compensate" with more assumptions.

§1 The Choiceless Polynomial Time Logic Presented

Here we present the logic. How does we "compute" when a model M satisfies a "sentence" θ? Note that the computation should not take too long and use too much "space" (in the "main" case: polynomial).

Informally, we start with a model M with each element an atom=ure-element, we successively define $N_t[M]$ and $N^+_t[M]$, an expansion of $N_t[M], t$ running on the stages of the "computation"; to get $N_{t+1}[M]$ from $N_t[M]$ we add few families of subsets of N_t, each family for some ψ consist of those defined by a formula $\psi(-, \bar{a})$ for some $\bar{a}$ from $N_t[M]$, and we update few relations or functions, by defining them from those of the previous stage. Those are $P_{t,\ell}$ for $\ell < m_1$. We may then check if a target condition holds, then we stop getting an answer: the model satisfies θ or fails by checking if some sentence χ holds; the natural target condition is when we stop to change. Note that each stage increases the size of $N_t[M]$ by at most a (fixed) power, however in $\|M\|$ steps we may have constructed a model of size $2^{\|M\|}$ but this is against our intentions. So we shall have a function $\mathbf{t}$ in $\|M\|$, normally polynomial, whose role is that when we have wasted too much resources (e.g. $\|N_t[M]\| + t$) we should stop the computation even if the target condition has not occurred, in this case we still have to decide what to do.

This involves some parameters. First a logic $\mathscr{L}$ telling us which formulas are allowable

- (a) in the inductive step (the $\psi_\ell(-, \bar{y}) - s$)
- (b) in stating "the target condition" (we use standard ones: $N^+_{t+1} = N^+_t$ or $P_0 = P_2$ or $c_0 = c_2$) and the χ telling us if the answer is yes or no.

Second, a family $\mathbf{T}$ of functions $\mathbf{t}$ (with domain the family of finite models) which tell us when we should stop having arrived to the limit of allowable resources (clearly the normal family $\mathbf{T}$ is $\{\mathbf{t}_k : k < \omega\}$ where $\mathbf{t}_k(M) = \|M\|^k$). So for Υ which describes the induction step (essentially how to get $N_{t+1}[M]$ from $N_t[M]$), χ telling us the answer and $\mathbf{t} \in \mathbf{T}$ we have a sentence $\theta_{\Upsilon,\chi,\mathbf{t}}$. We still have some variants, getting a logic $\mathscr{L}^{\mathbf{T}}_{\iota}[\mathscr{L}^*]$ for each $\iota \in \{1,2,3,4,5,6,7,11,22\}$.

In the case $\iota = 1$ we ignore $\mathbf{t}$, so $M \models \theta_{\Upsilon,\chi,\mathbf{t}}$ iff for some $t < \infty$ the target condition $N^+_t = N^+_{t+1}$ holds, or let $t = \infty$ and for this t, χ is satisfied by N^+_t. This is a very smooth definition, but we have lost our main goal. We may restrict ourselves to "good" sentences for which we always stop in "reasonable" time and space.

In the case $\iota = 2$, possibly we stop because of $\mathbf{t}$ before the target condition holds; in this case we say "$\theta_{\Upsilon,\chi,\mathbf{t}}$ is undefined for M". The case $\iota = 3$ is like $\iota = 2$ but we restrict ourselves to the so called "standard $\mathbf{T}$", where in $N_t[M]$ we have the natural numbers $< t$, so we can ignore the "time" as a resource as always $\|N_t[M]\| \geq t$. The case $\iota = 4$, is like $\iota = 3$ but instead stopping when $\|N_t[M]\|$ is too large, we stop when at least one of the families of sets added to $N_t[M]$ to form $N_{t+1}[M]$ is too large. For $\iota = 5$, is like the case $\iota = 2$, but an additional reason for stopping is $t > \mathbf{t}(\|M\|)$. The case $\iota = 6$ is as the case $\iota = 2$ separating the bounds on "space" (that is $\|N_t[M]\|$) and "time" (that is t), the case $\iota = 7$ is similar, not stopping for $N^+_t = N^+_{t+1}$. The cases $\iota = 11, \iota = 22$ are like $\iota = 1, \iota = 2$ respectively, but using $N_t[M, \Upsilon, \mathbf{t}]$ (see Definition 1.1(c)) instead $N_t[M, \Upsilon]$ but for $\iota = 22$ we separate $\mathbf{t}$ to two functions.

We treat as our main case $\iota = 3$, see more 1.7.

More formally

1.1 Definition. 1) We are given a model M, with vocabulary $\tau = \tau_{[0]}, \tau$ finite and $\in$ not in τ, let $\tau^+ = \tau_{[1]} = \tau \cup \{\in\}$. Considering the elements of M as atoms = urelements, we define $V_t[M]$ by induction of t : $V_0[M] = (M, \in\restriction M)$ and clearly with $\in\restriction M$ being empty (as we consider the members of M as atoms = "urelements"). Next $V_{t+1}[M]$ is the model with universe $V_t[M] \cup \{a : a \subseteq V_t[M]\}$ (by our assumption on "urelements" we have $a \subseteq V_t[M] \Rightarrow a \notin M$) with the predicates and individual constants and function symbols of τ interpreted as in M (so function symbols in τ are always interpreted as partial functions) and $\in^{V_{t+1}[M]}$ is $\in\restriction V_{t+1}[M]$.

2)

(A) We say $\Upsilon = (\bar{\psi}, \bar{\varphi}, \bar{P})$ is an inductive scheme for the logic $\mathscr{L}_{\text{f.o.}}$ or the language $\mathscr{L}_{\text{f.o.}}(\tau)$ (where $\mathscr{L}_{\text{f.o.}}$ is first order logic) if: letting $m_0 = \ell g(\bar{\psi}), m_1 = \ell g(\bar{\varphi})$ and $\tau_{[2]} = \tau_{[2]}[\Upsilon] = \tau_{[1]} \cup \{P_k : k < m_1\}$ we have

(a) $\bar{P} = \langle P_k : k < m_1 \rangle$ is a sequence (with no repetitions) of predicates and function symbols not in $\tau_{[1]}$ (notationally we treat an n-place function symbol as $(n+1)$-predicate); where P_k is an $\mathbf{m}^{\Upsilon}(P_k)$-place predicate. Let $\mathbf{m}^{\Upsilon}_1$ be the function giving this information and whether P_k is a predicate or a function symbol (so have domain $\{0, \ldots, m_1 - 1\}$)

(b) $\bar{\psi} = \langle \psi_\ell : \ell < m_0 \rangle, \psi_\ell = \psi_\ell(x; \bar{y}_\ell)$ is first order in the vocabulary $\tau_{[2]}$,

(c) $\bar{\varphi} = \langle \varphi_k : k < m_1 \rangle, \varphi_k = \varphi_k(\bar{x}_k)$ is first order in the vocabulary τ_2 with $\ell g(\bar{x}) = \mathbf{m}(P_k)$, moreover $\bar{x} = \langle x_i : i < \mathbf{m}(P_k) \rangle$.

(B) We say Υ is simple if

(d) each P_k is unary predicate and and each $\varphi_k(x)$ appears among the ψ_ℓ's (with empty $\bar{y}_\ell$) and for every hereditary model $N^* \subseteq \mathbf{V}_\infty[M]$, and a $\tau_{[2]}$–expansion N^+ of N^* we have $\{a : N^+ \models \varphi_\ell(a)\}$ is a member of N^* or is $\emptyset$

(C) We may write $m_0^\Upsilon, m_1^\Upsilon, \psi_\ell^\Upsilon, \mathbf{m}^\Upsilon, \mathbf{m}_1^\Upsilon, \varphi_\ell^\Upsilon, \psi_\ell^\Upsilon, P_\ell^\Upsilon$. We let

$$\Psi^\Upsilon = \{\psi_\ell(x, \bar{y}_\ell) : \ell < m_0^\Upsilon\}.$$

(D) We say Υ is predicative if each P_k is a predicate; we may restrict ourselves to this case for the general theorems. We say Υ is pure if $m_1 = 0$

(E) Υ is monotonic if $y \in x$ is ψ_ℓ for some $\ell < m_0^\Upsilon$ (this will cause N_t below to grow); no big loss if we restrict ourselves to such Υ. It is strongly monotonic if in addition each $\varphi_k(\bar{x})$ has the form $P_k(\bar{x}) \vee \varphi'_k(\bar{x})$ (this will cause also each P_k to grow)

(F) Υ is i.c. if each P_k is (informally an individual constants scheme) a zero place function symbol; in this case if P_k is well defined we may write it as c_k.

3) For $M, \tau = \tau_{[0]}, \tau_{[1]}, \tau_{[2]}$, and $\Upsilon = (\bar{\psi}, \bar{\varphi}, \bar{P})$ as above, we shall define by induction on t a submodel $N_t = N_t[M]$ of $V_t[M]$ and $\bar{P}_t = \langle P_{t,k} : k < m_1 \rangle$ and $N_t^+[M]$ and $\mathscr{P}_{t,\ell}$ (for $\ell < m_0$) as follows; more exactly we are defining $N_t[M, \Upsilon], P_{t,k}[M, \Upsilon]$ for $k < m_1, \mathscr{P}_{t,k}[M, \Upsilon]$ for $k < m_0$.

We let $N_t^+[M, \Upsilon] = (N_t[M, \Upsilon], P_{t,0}[M, \Upsilon], \ldots, P_{t,m_1-1}[M, \Upsilon])$.

Case 1: $t = 0$: $N_t[M] = V_0[M]$ and $P_{t,k} = \emptyset$ (an $\mathbf{m}^\Upsilon(k)$-place relation).

Case 2: $t + 1$: $N_{t+1}[M]$ is the submodel of $V_{t+1}[M]$ with set of elements the transitive closure of $M \cup \bigcup_{\ell < m_0} \mathscr{P}_{t,\ell}[M]$ where we define $\mathscr{P}_{t,k}[M]$ and $P_{t+1,\ell}$ by:

$$\mathscr{P}_{t,\ell}[M] = \left\{ \{a \in N_t[M] : N_t^+[M] \models \psi_\ell(a, \bar{b})\} : \bar{b} \in {}^{(\ell g(\bar{y}_i))}(N_t[M]) \right\}$$

$$P_{t,k} = \{\bar{a} \in {}^{\mathbf{m}(k)}(N_t[M]) : N_t^+[M] \models \varphi_k[\bar{a}]\}$$

but if P_k is a function symbol,

$$P_{t,\ell} = \{\bar{a}^\frown\langle b \rangle \in N_t[M] : N_t^+[M] \models \varphi_\ell(\bar{a}, b) \ \& \ (\exists! y)\varphi_\ell(\bar{a}, y)\}$$

(so if Υ is simple, then for $t > 1$, $\psi_k(x, \bar{y}) \in \mathscr{L}_{\rm f.o.}(\tau_{[2]})$ is actually $\psi'_k(x, \bar{y}, \bar{c}_{t-1}) \in \mathscr{L}_{\rm f.o.}(\tau_{[1]})$).

Case 3: $t = \infty$

$$N_t = \bigcup_{s<t} N_s, P_{t,k} = \bigcup_{s<t} P_{s,k}.$$

3A) If in addition $\mathbf{t} \in \mathbf{T}$ and Υ is monotonic (see part (2) clause (E)) we define $N_t[M, \Upsilon, \mathbf{t}], P_{t,\ell}[M, \Upsilon, \mathbf{t}]$ and $\mathscr{P}_{t,\ell}[M, \Upsilon, \mathbf{t}]$ as in part (3) except that the universe of $N_{t+1}[M, \Upsilon, \mathbf{t}]$ is the transitive closure of $M \cup \bigcup\{\mathscr{P}_{t,\ell}[M, \Upsilon, \mathbf{t}] : \ell < m_0$ and $\mathscr{P}_{t,\ell}[M, \Upsilon, \mathbf{t}]$ has at most $\mathbf{t}(\|M\|)$ members$\}$.
4) We say Υ is standard if some ψ_i guarantees that $t \subseteq N_t[M]$ (so a natural number s belongs to $N_t[M]$ iff $s < t$; remember that we identify the natural number t with the set $\{0, 1, \ldots, t-1\}$).
5) Let q.d.(φ) be the quantifier depth of the formula φ.
6) We may replace above first order logic by another logic $\mathscr{L}$. We let $\mathscr{L}_{\rm f.o.}$ denote first order, $\mathscr{L}_{\rm card}$ is defined just like first order logic except that we demand that Υ is standard and defining inductively what is a formula, we allow the formation of formulas the form $|\{x : \theta(x, \bar{y})\}| = s$. We let $\mathscr{L}_{{\rm card},\mathbf{T}}$ (on $\mathbf{T}$ see below) be defined just like $\mathscr{L}_{\rm f.o.}$ but for each $\mathbf{t} \in T$ we allow the quantifier $(Q_{\mathbf{t}}x)\varphi(x, \bar{y})$ with

$$N \models (Q_{\mathbf{t}}x)\varphi(x, \bar{a}) \quad \text{iff} \quad \mathbf{t}(|{\rm ure}(N)|) < |\{b : N \models \varphi(b, \bar{a}]\}|,$$

where ure(N) is the set of urelements of N.
6A) Lastly, let $\mathscr{L}_{\rm f.o.+na}$ be like $\mathscr{L}_{\rm f.o.}$ but we add one atomic formula $|{\rm atoms}| = x$ being interpreted as: the number of atoms is x. So this can be expressed in $\mathscr{L}_{{\rm card},\mathbf{T}}$ where $\mathbf{T} = \{{\rm id}\}, {\rm id}(n) = n$.

1.2 Remark. Alternatively to $\mathscr{L}_{\rm card}$: have a quantifier

$$(Q^{\rm eq}x_1, x_2)(\varphi_2(x_1, \bar{y}_1), \varphi_2(x_2, \bar{y}_2)),$$

which says that $|\{x : \varphi_2(x, \bar{y}_1)\}| = |\{x : \varphi_2(x, \bar{y}_2)\}|$.

In the definition below the reader can concentrate on $\iota = 3$. The "$t \geq 2$ & ..." is not a serious matter.

1.3 Definition. Let $\mathbf{T}$ be a set of functions $\mathbf{t} : \mathbb{N} \to \mathbb{N} \cup \{\infty\}$ and $\mathscr{L}^*$ be a logic ($\mathscr{L}_{\rm f.o.}$ or $\mathscr{L}_{\rm f.o.+na}$ or $\mathscr{L}_{\rm card}$ usually) and let τ be a vocabulary. If $\mathbf{t}$ is constantly ∞ we may write ∞.

We define for $\iota = 1, 2, 3, 4, 5, 6, 7, 11, 22$ the logic $\mathscr{L}^{\mathbf{T}}_\iota[\mathscr{L}^*]$ below. For all of those logics the set of sentences for a vocabulary τ called $\mathscr{L}^{\mathbf{T}}_\iota[\mathscr{L}^*](\tau)$ is a subset of $\Theta = \Theta_\tau = \Theta_\tau[\mathscr{L}^*, \mathbf{T}] = \{\theta_{\Upsilon,\chi,\mathbf{t}} : \Upsilon$ an inductive scheme for $\mathscr{L}^*(\tau), \chi \in \mathscr{L}^*(\tau)$ and $\mathbf{t} \in \mathbf{T}\}$, (equal if not said otherwise). Also for most of those logics we define the stopping time $t_\iota[M, \Upsilon, \mathbf{t}]$ or $t_\iota[M, \Upsilon]$ (if $\mathbf{t}$ does not matter). The satisfaction relation for $\mathscr{L}^{\mathbf{T}}_\iota[\mathscr{L}^*](\tau)$ is denoted by $\models_\iota$. Also we write $\theta_{\Upsilon,\chi}$ instead $\theta_{\Upsilon,\chi,\mathbf{t}}$ if $\mathbf{t}$ does not matter. (We may let Dom$(\mathbf{t})$ be the set of relevant structures, see 0.1).

Case 1: $\iota = 1$.

We let

$$t_\iota[M, \Upsilon] = \ \mathrm{Min}\{t : t \geq 2 \text{ and } N^+_t[M, \Upsilon] = N^+_{t+1}[M, \Upsilon]\}.$$

(If there is no such $t \in \mathbb{N}$ we let it be ∞ (i.e., ω for set theorists) and we could also have used "undefined"; note that $\mathbf{t}$ does not appear).

$M \models_\iota \theta_{\Upsilon,\chi}$ iff $N^+_t[M, \Upsilon] \models \chi$ for $t = t_\iota[M, \Upsilon]$.

Case 2: $\iota = 2$.

We let $t_\iota[M, \Upsilon, \mathbf{t}] = \ \mathrm{Min}\{t : \|N_{t+1}[M, \Upsilon]\| + (t+1) > \mathbf{t}(\|M\|)$ or $t \geq 2 \ \ \& \ N^+_t[M, \Upsilon] = N^+_{t+1}[M, \Upsilon]\}$ and

(a) if for $t = t_\iota[M, \Upsilon, \mathbf{t}]$ we have $t \geq 2 \ \ \& \ \ N^+_t[M, \Upsilon] = N^+_{t+1}[M, \Upsilon]$ then $\theta_{\Upsilon,\chi,\mathbf{t}}$ is true or false in M iff $N^+_t[M, \Upsilon] \models \chi$ or $N^+_t[M, \Upsilon] \models \neg\chi$ respectively and we write $M \models_\iota \theta_{\Upsilon,\chi,\mathbf{t}}$ or $M \models_\iota \neg\theta_{\Upsilon,\chi,\mathbf{t}}$ respectively, (so $\neg\theta_{\Upsilon,\chi,\mathbf{t}}$ is equivalent to $\theta_{\Upsilon,\neg\chi,\mathbf{t}}$).

(b) if $t_\iota[M, \Upsilon, t] = \mathbf{t}(\|M\|) + 1$ we say "$M \models_\iota \theta_{\Upsilon,\chi,t}$ is undefined" and we say "the truth value of $\theta_{\Upsilon,\chi,\mathbf{t}}$ in M is undefined".

Case 3: $\iota = 3$.

As in Case 2 but we restrict ourselves to standard Υ, see Definition 1.1(4), and let $t_\iota[M, \Upsilon, \mathbf{t}] = \ \mathrm{Min}\{t : \|N_{t+1}[M, \Upsilon]\| > \mathbf{t}(\|M\|)$ or $t \geq 2 \ \ \& \ \ N^+_t[M, \Upsilon] = N^+_{t+1}[M, \Upsilon]\}$ and define $\models_\iota$ as in Case 2.

Case 4: $\iota = 4$.

As in Case 2 but we restrict ourselves to standard Υ and let:

$$\begin{aligned} t_\iota[M, \Upsilon, \mathbf{t}] = \ \mathrm{Min}\{t :& \text{for some } k < m_0 \text{ the set } \mathscr{P}_{t+1,k}[M, \Upsilon] \\ & \text{has } > \mathbf{t}(\|M\|) \text{ members or} \\ & t \geq 2 \ \ \& \ \ N^+_t[M, \Upsilon] = N^+_{t+1}[M, \Upsilon]\} \end{aligned}$$

(so it can be ∞; but by a choice of e.g. φ_0 we can guarantee $\mathscr{P}_{t,0} = \{0, \ldots, t-1\}$ so that this never happens) and define $\models_\iota$ as in Case 2.

Case 5: $\iota = 5$.

As in Case 2 but we restrict ourselves to standard Υ and $t_\iota[M, \Upsilon, \mathbf{t}] = \ \mathrm{Min}\{t : t > \mathbf{t}(\|M\|)$ or for some $k < m_0$ the set $\mathscr{P}_{t+1,k}[M, \Upsilon]$ has $> \mathbf{t}(\|M\|)$ members or $t > 2 \ \ \& \ \ N^+_t[M, \Upsilon] = N^+_{t+1}[M, \Upsilon, \mathbf{t}]\}$.

Case 6: $\iota = 6$.

As in the case $\iota = 2$ but

$$\begin{aligned} t_\iota[M, \Upsilon, \mathbf{t}] = \ \mathrm{Min}\{t : t > \mathbf{t}^{tm}(\|M\|) &\text{ or } \|N_t[M, \Upsilon]\| > \mathbf{t}^{\mathrm{sp}}(\|M\|) \\ &\text{or } N^+_t[M, \Upsilon] = N^+_{t+1}[M, \Upsilon]\} \end{aligned}$$

where

$$\mathbf{t}^{\rm tm}(n) = \mathbf{t}(2n), \mathbf{t}^{\rm sp}(n) = \mathbf{t}(2n+1)$$

and, of course, *tm*, *sp* stand for time and space, respectively. So we may replace $\mathbf{t}$ by two functions $\mathbf{t}^{\rm tm}, \mathbf{t}^{\rm sp}$ and write sentences as $\theta_{\Upsilon,\chi,\mathbf{t}',\mathbf{t}''}$ (similarly for $\iota = 7, 22$).

Case 7: $\iota = 7$.

We define $t_\iota[M, \Upsilon, \mathbf{t}] =$ Min$\{t : t > \mathbf{t}^{\rm tm}(\|M\|)$ or $\|N_t[M, \Upsilon]\| > \mathbf{t}^{\rm sp}(\|M\|)$ or ≥ 2 & $N_t^+[M, \Upsilon] = N_{t+1}^+[M, \Upsilon]\}$

and let:

$$M \models_\iota \theta_{\Upsilon,\chi,\mathbf{t}} \text{ iff } N_t[M, \Upsilon] \models \chi \text{ for } t = t_\iota[M, \Upsilon].$$

(so unlike cases 2-6, the truth value is always defined)

Case 11: $\iota = 11$.

As in the case $\iota = 1$, but we use $N_t[M, \Upsilon, \mathbf{t}], \bar{P}_t[M, \Upsilon, \mathbf{t}]$ (see Definition 1.1(3A)).

Case 22: $\iota = 22$.

As in the case $\iota = 2$, but we use $N_t[M, \Upsilon, \mathbf{t}^{\rm wd}], \bar{P}_t[M, \Upsilon, \mathbf{t}^{\rm wd}]$ where $\mathbf{t}^{\rm wd} \in \mathbf{T}$ is defined by $\mathbf{t}^{\rm wd}(n) = \mathbf{t}(2n)$ (wd for width) and define t_ι by

$$\begin{aligned} t_\iota[M, \Upsilon, \mathbf{t}] = \text{Min}\{t :& N_{t+1}[M, \Upsilon, \mathbf{t}^{\rm wd}] + (t+1) > \mathbf{t}^{\rm ht}(\|M\|) \text{ or} \\ & t \geq 2 \ \& \ N_t^+[M, \Upsilon, \mathbf{t}^{\rm wd}] = N_{t+1}^+[M, \Upsilon, \mathbf{t}^{\rm wd}]\} \end{aligned}$$

where (ht for height)

$$\mathbf{t}^{\rm ht} \in \mathbf{T} \text{ is defined by } \mathbf{t}^{\rm ht}(n) = \mathbf{t}(2n+1).$$

We may write $\mathbf{t}^{\rm wd}, \mathbf{t}^{\rm ht}$ instead of $\mathbf{t}$.

1.4 Remark. Alternatively:

Case 10 + ι: For $\iota = 1, \ldots, 7$.

Like the case ι but we use $N_t[M, \Upsilon, \mathbf{t}], \bar{P}_t[M, \Upsilon, \mathbf{t}]$ and let $\mathbf{t}_{10+\iota} = \mathbf{t}_\iota$.

Case 20 + ι: $\iota = 1, \ldots, 7$.

As in the cases $\iota = 1, \ldots, 7$, but we use $N_t[M, \Upsilon, \mathbf{t}^{\rm wd}], \bar{P}_t[M, \Upsilon, \mathbf{t}^{\rm wd}]$, where $\mathbf{t}^{\rm wd} \in \mathbf{T}$ is defined by $\mathbf{t}(n) = \mathbf{t}(2n)$ (wd for width) and we replace $\mathbf{t}$ by $\mathbf{t}^{\rm ht}, \mathbf{t}^{\rm ht}(n) = \mathbf{t}(2n+1)$, where for $x =$ tm,sp we derive from $\mathbf{t}^{\rm ht}$ the functions $\mathbf{t}^{{\rm ht},x}_{20+\iota} = (\mathbf{t}^{\rm ht})^x$.

1.5 Definition. 1) In Definition 1.3 we say "$\theta_{\Upsilon,\chi,\mathbf{t}}$ is ι-good" when: for every finite model M one of the following occurs:

- (a) $\iota = 1$ and $t_\iota[M,\Upsilon] < \infty$
- (b) $\iota \in \{2,3,4,5,6,7\}$ and in the definition of $t_\iota[M,\Upsilon,\mathbf{t}]$ always the last possibility occurs and not any of the previous ones
- (c) $\iota = 11$ as in the case $\iota = 1$ using $N_t[M,\Upsilon,\mathbf{t}], \bar{P}_t[M,\Upsilon,\mathbf{t}]$
- (d) $\iota = 22$, as for $\iota = 2$ using $N_t[M,\Upsilon,\mathbf{t}], \bar{P}_t[M,\Upsilon,\mathbf{t}]$.

2) Let $\mathscr{L}^{\mathbf{T}}_\iota(\mathscr{L}^*)^{\rm good}$ be the logic $\mathscr{L}^{\mathbf{T}}_\iota(\mathscr{L}^*)$ restricted to ι-good sentences.
3) If in $\theta_{\Upsilon,\chi,\mathbf{t}}$ we omit χ we mean $P_{t,0}$ = the set of atoms.
4) We say that $M \models_\iota \theta_{\Upsilon,\chi,\mathbf{t}}$ is in a good way if this case of part (1) holds

1.6 Remark. We can replace in cases ι = 2,3,4 clause (b), the statement $N^+_t[M,\Upsilon] = N^+_{t+1}[M,\Upsilon]$ by a sentence χ_1.

1.7 Discussion: 0) Note that considering several versions should help to see how canonical is our logic.
1) The most smooth variant for our purpose is $\iota = 4$, and the most natural choice is $\mathscr{L}^* = \mathscr{L}_{\rm card}$ or $\mathscr{L}^* = L_{{\rm card},\mathbf{T}}$, but we are not less interested in the choice $\mathscr{L}^* = \mathscr{L}_{\rm f.o.}, \mathscr{L}^* = \mathscr{L}_{\rm f.o.+na}$. From considering the motivation the most natural $\mathbf{T}$ is $\{n^m : m < \omega\}$, and $\iota = 3$.
2) For e.g. $\iota = 1,2,3$ some properties of M can be "incidentally" expressed by the logic, as the stopping time gives us some information concerning cardinality. For example let Y be a complicated set of natural numbers, e.g. non-recursive, and let $\mathbf{t}^* \in \mathbf{T}$ be: $\mathbf{t}(\|M\|)$ is $\|M\| + 10$ if $\|M\| \in Y$ and $\mathbf{t}(\|M\|) = \|M\| + 6$ if $\|M\| \notin Y$. We can easily find $\theta = \theta_{\Upsilon,\chi,\mathbf{t}^*}$, with Υ a standard induction scheme such that it stops exactly for $t = 8$ and χ saying nothing (or if you like saying that there are 8 natural numbers). Clearly for $\iota = 2,3$ we have $M \models_\iota \theta_{\Upsilon,\chi,\mathbf{t}^*}$ if $\|M\| \in Y$ and "not $M \models_\iota \theta_{\Upsilon,\chi,\mathbf{t}^*}$" if $\|M\| \notin Y$. Of course, more generally, we could first compute some natural number from M and then compare it with $\mathbf{t}(\|M\|)$. This suggests preferring the option $\models_\iota$ undefined in clause (b) of case 2, Definition 1.3 rather than false.
3) If you like set theory, you can let t be any ordinal; but this is a side issue here; see end of §5.

Implicit in 1.3 (and an alternative to 1.3) is (note: an (M,Υ)-candidate $(N,\bar{P})$ is what looks like a possible $N_t[M,\Upsilon]$ and a $(\Upsilon,\mathbf{t})$-successor of it is what looks like $N_{t+1}[M,\Upsilon]$):

1.8 Definition. Let M,Υ as in Definition 1.3 be given.
1) We say $(N,\bar{P})$ is an M-candidate or (M,Υ)-candidate if:

- (a) N is a finite transitive submodel of $\bigcup\limits_t V_t[M]$ which includes M, expanded by the relations of M (so it is a $(\tau_M)_{[1]}$-model)

(b) $\bar{P} = \langle P_k : k < m_1 \rangle, P_k$ an $\mathbf{m}^{\Upsilon}(k)$-relation on N or a partial $(\mathbf{m}^{\Upsilon}(k) - 1)$-place function on N when P_k is a predicate or a function symbol respectively.

In fact here the only information from Υ used is $\mathbf{m}^{\Upsilon}_1$ of Υ, so we may write "$(M, \mathbf{m}^{\Upsilon}_1)$-candidate".
2) We say $(N', \bar{P}')$ is the $(\Upsilon, \mathbf{t})$-successor of $(N, \bar{P})$ if $(N', \bar{P}'), (N, \bar{P})$ satisfies what $N^+_{t+1}[M, \Upsilon, \mathbf{t}], N^+_t[M, \Upsilon, \mathbf{t}]$ satisfies in Definition 1.1(3A), so

$$|N'| = \text{ is the transitive closure of } M \cup \bigcup_{\ell < m_1} A_\ell[N, \Upsilon, \mathbf{t}],$$

where $A_\ell = A_\ell[N, \Upsilon, \mathbf{t}]$ is $\mathscr{P}_\ell[N, \Upsilon] = \big\{\{a : (N, \bar{c}) \models \psi_\ell(a, \bar{b})\} : \bar{b} \in {}^{\ell g(\bar{g}_\ell)}N\big\}$ if this family has $\leq \mathbf{t}(M)$ members or is equal to N and A_ℓ is empty otherwise.
2A) We say $(N', \bar{P})$ is the Υ-successor of $(N, \bar{P})$ if $(N', \bar{P}'), (N, \bar{P})$ satisfies what $N^+_{t+1}[M, \Upsilon], N^+_t[M, \Upsilon]$ satisfies in Definition 1.1(3); this means just that $(N', \bar{P}')$ is the (Υ, ∞)-successor of $(N, \bar{P})$.
2B) If Υ is pure (i.e. $m^{\Upsilon}_1 = 0$), actually only $\bar{\psi}^{\Upsilon}$ count and we may replace Υ by $\bar{\psi}^{\Upsilon}$.
2C) We say that $(N, \bar{P})$ is a $(M, \Upsilon)^+$-candidate if it is an (M, Υ)-candidate and the sets $\emptyset, |M|$ (= set of atoms) belongs to N.
3) We define $N_t = N_t[M, \Upsilon, \mathbf{t}]$ and $\bar{P}_t = \bar{P}_t[M, \Upsilon, \mathbf{t}]$ by induction on t as follows:

for $t = 0$ it is M (i.e. with $P_{t,k} = \emptyset$),
for $t + 1, (N_{t+1}, \bar{P}_{t+1})$ is the $(\Upsilon, \mathbf{t})$–successor of $(N_t, \bar{P}_t)$, see below 1.9,
for $t = \infty$ we take the union.

1.9 Claim. *1) If $(N, \bar{P})$ is an (M, Υ)-candidate, it has exactly one $(\Upsilon, \mathbf{t})$-successor (and exactly one Υ-successor).*
2) The pair $(N_t[M, \Upsilon, \infty], \bar{P}_t[M, \Upsilon, \infty])$ defined in Definition 1.8(3), is equal to the pair $(N_t[M, \Upsilon], \bar{P}_t[M, \Upsilon])$ defined in Definition 1.1(3).
3) If Υ is monotonic, $(N', \bar{P}')$ the Υ-successor of $(N, \bar{P})$ where both are $(M, \Upsilon)^+$-candidates, then $N \subseteq N'$; if Υ is also standard then $N \subset N'$.
4) If Υ is strongly monotonic (see Definition 1.1(2)(E)) and $(N', \bar{P}')$ is the Υ-successor of $(N, \bar{P})$ both are $(M, \Upsilon)^+$-candidates, then $N \subseteq N'$ and $P_\ell \subseteq P'_\ell$ for $\ell < m^{\Upsilon}_1$.

There are many obvious inclusions between the variants of logics by natural translations. We mention the following claim which tells us that there is no real harm if we restrict ourselves to pure Υ's.

1.10 Claim. *1) Assume the Υ is an inductive scheme in $\mathscr{L}_{\text{f.o.}}(\tau^+), \chi$ a sentence in $\mathscr{L}_{\text{f.o.}}(\tau^+)$. Then we can find a pure inductive scheme Υ^* in $\mathscr{L}_{\text{f.o.}}(\tau^+)$ and r^*, r^{**} and $\mathbf{p}^*$ and sentences θ^*, χ^*_1 and formulas $\varphi^*(x), \varphi^*_k(\bar{x}_k)$ for $k < m^{\Upsilon}_1, \ell g(\bar{x}_\ell) = \mathbf{m}^{\Upsilon}(k)$ in $\mathscr{L}_{\text{f.o.}}(\tau)$ such that:*

$\boxtimes$ *for every* τ*-model and* t *we have, if* $t^* = r^{**} + r^* t$ *then:*

- (a) *the set* $N_t[M, \Upsilon]$ *is* $\{a \in N_{t^*}[M, \Upsilon^*] : N_{t^*}[M, \Upsilon^*] \models \varphi^*[a]\}$,
- (b) $P_{t,k}[M, \Upsilon] = \{\bar{a} \in {}^m(N_{t^*}[M, \Upsilon^*]) : N_{t^*}[M, \Upsilon^*] \models \varphi^*_k[\bar{a}]\}$, *where* $m = \mathbf{m}^\Upsilon(k)$,
- (c) $N_t[M, \Upsilon] \models \chi$ *iff* $N_{t^*}[M, \Upsilon^*] \models \chi^*$,
- (d) $N^+_t[M, \Upsilon] = N^+_{t+1}[M, \Upsilon]$ *(i.e. it stops) iff* $N_{t^*}[M, \Upsilon^*] \models \theta^*$ *iff* $N_{t^*}[M, \Upsilon^*] = N_{t^*+1}[M, \Upsilon^*]$,
- (e) $N_{t^*}[M, \Upsilon^*]$ *has exactly* $\mathbf{p}^*(\|N_t[M, \Upsilon]\|)$ *elements, and* $N_{t^*+r}[M, \Upsilon^*]$ *has* $\leq \mathbf{p}^*(\|N_t[M, \Upsilon]\|)$ *when* $r < r^*$ *and* $\mathbf{p}^*$ *an integer polynomial,*
- (f) *if* Υ *is standard then so is* Υ^*.

2) Similarly using a logic $\mathscr{L}^*$ *if it is closed under first order operations and substitutions.*

Remark. We can similarly deal with $N_t[M, \Upsilon, \mathbf{t}]$, but then we have to deal with some form of cardinality quantifiers, etc.

Proof. For simplicity we assume that Υ is standard. Now for every $(M, \mathbf{m}^\Upsilon_1)$–candidate $(N, \bar{P})$ we shall define a M-candidate $N^* = N^*_{(N,\bar{P})}$. We shall have $N^+_{r^{**}+r^*t}[M, \Upsilon^*]$ is $N^*_{N^+_t[M,\Upsilon]}$.

The set of natural numbers of N^* is $\{s : s < r^{**} + r^* t\}$. The universe of N^* is the union of the following sets:

- (a) N
- (b) $\{\{x, r^*t\} : x \in N\}$
  (used to define N),
  let $a_{x,k,m} =: \{x, r^*t + 1 + k, r^*t + 1 + m^\Upsilon_1 + m\}$ for $x \in N, k < m^\Upsilon_1, m < \mathbf{m}^\Upsilon(k)$
  (used to help to code $P_{t,k}$)
- (c) $\{\{a_{x_m,k,m} : m < \mathbf{m}^\Upsilon(k)\} \cup \{r^*t + 1 + k, r^*t + 1 + m^\Upsilon_1 + i\} :$
  $k < m^\Upsilon_1, x_0, \ldots, x_{\mathbf{m}^\Upsilon(k)-1} \in N$ and $i \in \{0, 1\}$ and
  $i = 1 \Leftrightarrow \langle x_m : m \in \mathbf{m}^\Upsilon(k)\rangle \in P_{t,k}\}$
- (d) some more elements to take care of the

  " $N_{t^*}[M, \Upsilon^*]$ has exactly $\mathbf{p}^*(\|N_t[M, \Upsilon]\|)$ elements "

  (if we agree to $N_{t^*}[M, \Upsilon^*]$ has exactly $\mathbf{p}(\|N_t[M, \Upsilon^*], t\|)$ then this is not necessary).

The rest should be clear. $\square_{1.10}$

We try to sort out some of the relations between these logics by checking when two variants of a sentence say related things, for quite many ι_1, ι_2.

1.11 Claim. *Let* $\iota_1, \iota_2 \in \{1, \ldots, 7\}$ *and* $\theta_\ell = \theta_{\Upsilon,\chi,\mathbf{t}_\ell} \in \mathscr{L}^{\mathbf{T}}(\mathscr{L}^*)(\tau)$ *for* $\ell = 1, 2$ *and consider*

- (α) θ_2 *is* ι_2*-good implies* θ_1 *is* ι_1*-good,*
- (β) *for every (finite)* τ*-model* $M, (M \models_{\iota_2} \theta_2) \Rightarrow (M \models_{\iota_1} \theta_1)$,
- $(\beta)^-$ *if* θ_ℓ *is* ι_ℓ*-good for* $\ell = 1, 2$ *then* $M \models_{\iota_2} \theta_2 \Rightarrow M \models_{\iota_1} \theta_1$.

In the following clauses we list cases which holds under various conditions.

- $(A)(\alpha) + (\beta)$ *if* $\iota_1 = \iota_2$ *and* $\mathbf{t}_1 < \mathbf{t}_2$,
- $(B)(\alpha) + (\beta)$ *if* $\iota_1 = 2, \iota_2 = 3, \Upsilon$ *is standard and* $\mathbf{t}_1 \geq 2\mathbf{t}_2$,
- $(C)(\alpha) + (\beta)$ *if* $\iota_1 = 3, \iota_2 = 2, \Upsilon$ *is standard and* $\mathbf{t}_1 \geq \mathbf{t}_2$,
- $(D)(\alpha) + (\beta)$ *if* $\iota_1 = 4, \iota_2 = 3, \Upsilon$ *is standard and* $\mathbf{t}_1 \geq \mathbf{t}_2$,
- $(E)(\alpha) + (\beta)^-$ *if* $\iota_1 = 3, \iota_2 = 4, \Upsilon$ *is standard and* $\mathbf{t}_1$ *is large enough,*
- $(F)(\alpha) + (\beta)$ *if* $\iota_1 = 5, \iota_2 = 3, \Upsilon$ *is standard and* $\mathbf{t}_1 \geq \mathbf{t}_2$,
- $(G)(\alpha)+(\beta)$ *if* $\iota_1 = 3, \iota_2 = 5, \Upsilon$ *is standard and* $(\forall n)[\mathbf{t}_1(n) \geq n + m_0^\Upsilon \mathbf{t}_2(n)\mathbf{t}_2(n)]$,
- $(H)(\alpha) + (\beta)^-$ *if* $\iota_1 = 1, \iota_2 = 2$,
- $(I)(\alpha) + (\beta)^-$ *if* $\iota_1 = 2, \iota_2 = 1$ *and* $\mathbf{t}_1$ *is large enough,*
  (i.e. $\mathbf{t}_1(n) > \mathrm{Max}\{t_{\iota_2}[M, \Upsilon, \mathbf{t}_2] : M$ *a* τ*-model with universe* $[n]$ *and* $t_{\iota_2}[M, \Upsilon, \mathbf{t}_2] < \infty\}$*; note that* Max *is taken on a finite set)*
- $(J)(\alpha) + (\beta)$ *if* $\iota_1 = 6, \iota_2 = 2$ *and* $\mathbf{t}_1(2n) \geq \mathbf{t}_2(n), \mathbf{t}_1(2n+1) \geq \mathbf{t}_2(n)$,
- $(K)(\alpha) + (\beta)$ *if* $\iota_1 = 2, \iota_2 = 6$ *and* $\mathbf{t}_2 \geq \mathbf{t}_1(2n) + \mathbf{t}_1(2n+1)$,
- $(L)(\alpha) + (\beta)$ *if* $\iota_1 = 7, \iota_2 = 6$ *and* $\mathbf{t}_1 \geq \mathbf{t}_2$
  (note: after "good" stopping, nothing changes) $\mathbf{t}_1$ *is large enough*

Proof. Straightforward.

1.12 Conclusion. 1) Assume that $\mathbf{T}$ satisfies

- $(*)$ $(\forall \mathbf{s} \in \mathbf{T})(\forall m)(\exists \mathbf{t} \in \mathbf{T})(\forall n)(\mathbf{t}(n) \geq n + m(\mathbf{s}(n))^2)$.

Then the logics $\mathscr{L}_\iota^{\mathbf{T}}(\mathscr{L}^*)^{\mathrm{good}}$ for $\iota = 2, 3, 4, 5, 6$ are weakly equivalent where $\mathscr{L}^1, \mathscr{L}^2$ are weakly equivalent if $\mathscr{L}^1 \leq_{\mathrm{wk}} \mathscr{L}^2$ and $\mathscr{L}^2 \leq_{\mathrm{wk}} \mathscr{L}^1$; where $\mathscr{L}^1 \leq_{\mathrm{wk}} \mathscr{L}^2$ means that for every sentence $\theta_1 \in \mathscr{L}^1$ there is $\theta_2 \in \mathscr{L}^2$ such that for every M we have[1] $M \models \theta_1$ implies $M \models \theta_2$

2) If in addition $\mathbf{T}$ consists of integer polynomials and $\mathscr{L}_* \in \{\mathscr{L}_{\mathrm{f.o.+na}}, \mathscr{L}_{\mathrm{card}}\}$ we can add $\iota = 7$.

1.13 Remark. In part 1.12(2) we can replace the assumption on $\mathbf{t}$ demanding only that:

[1] Note that if teh truth value of θ_1 in M is undefined, then the implication is trivial.

(∗) for every $\mathbf{t} \in \mathbf{T}$ view $\mathbf{t}$ as $(\mathbf{t}^{\rm tm}, \mathbf{t}^{\rm sp})$ there is $\mathbf{s} \in \mathbf{T}$ such that we can "compute" $\mathbf{t}^{\rm tm}(n), \mathbf{t}^{\rm sp}(n)$ is $\mathbf{s}^{\rm tm}(n)$ time, $\mathbf{s}^{\rm sp}(n)$ space in the relevant sense (using the given $\mathscr{L}_*$, etc.).

1.14 Claim. *The family of good sentences in the logic $\mathscr{L}^{\mathbf{T}}_{\iota}(\mathscr{L}^*_{\rm f.o.})$, that is the logic $\mathscr{L}^{\mathbf{T}}_{\iota}(\mathscr{L}^*_{\rm f.o.})^{\rm good}$, is closed under the following: the Boolean operation, $(\exists x)$, and substitution (that is, up to equivalence) when at least one of the following holds*

$(*)_1$ $\iota = 1$

$(*)_2$ $\iota \in \{2,3,5\}$ *and* $\mathbf{T}$ *satisfies* $(*)$ *of 1.12*

$(*)_3$ $\iota = 11$

$(*)_4$ $\iota = 22$ *and for each* $\mathbf{t} \in \mathbf{T}, \mathbf{T}[\mathbf{t}^{\rm wd}] =: \{\mathbf{s}^{\rm ht} : \mathbf{s} \in \mathbf{T}, \mathbf{s}^{\rm wd} = \mathbf{t}^{\rm wd}\}$ *satisfies* $(*)$ *of 1.12.*

Proof. Straight.

§2 The General Systems of Partial Isomorphisms

Though usually our aim is to compare two models M_1, M_2 we first concentrate on one model M; this, of course, gives shorter definitions.

Our aim is to have a family $\mathscr{F}$ of partial automorphisms as in Ehrenfeucht-Fraïssé games (actually Karp), of the model M we analyze, not total automorphism which is too restrictive. But this family has to be lifted to the N_t's. Hence their domains (and ranges) may and should sometimes contain an element of high rank. It is natural to extend each $f \in \mathscr{F}$ to $G_t(f)$, a partial automorphism of N_t. So we should not lose anything when we get up on t. The solution is $I \subseteq \{A : A \subseteq M\}$ closed downward and $\mathscr{F}$ (could have used $\langle \mathscr{F}_\ell : \ell < m_1 \rangle$), a family of partial automorphisms of M. So every $x \in N_t$ will have a support $A \in I$ and for $f \in \mathscr{F}$, its action on A determines its action on x, $(G_t(f)(x)$ in this section notation). It is not unreasonable to demand that there is the smallest support, still this demand is somewhat restrictive (or we have to add imaginary elements as in [Sh:a] or [Sh:c], not a very appetizing choice here).

But how come we in stage $t+1$ succeed to add "all sets $X = X_{\ell,\bar{b}}$" definable by $\psi_\ell(x, \bar{b})$ for some sequence $\bar{b} \in {}^{\ell g(\bar{b})}N_t$? Let m be such that $\bar{b} = \langle b_1, \ldots, b_m \rangle$.

The parameters $b_1, \ldots, b_m$ each has a support say $A_1, \ldots, A_m$ resp., all in I; so when we have enough mappings in the family $\mathscr{F}$, the new set has in some sense the support $A = \bigcup_{\ell=1}^{m} A_\ell$, in the sense that suitable partial mappings act as expected. So if $y \in N_t$ has support B (BRy in this section notation), $f \in \mathscr{F}, A \cup B \subseteq {\rm Dom}(f)$ and $f \restriction A = {\rm id}_A$, then the mapping $G_t(f)$ which f induces in N_t will satisfy $y \in X_{\ell,\bar{b}} \Leftrightarrow (G(f))(y) \in X_{i,\bar{b}}$.

But we are not allowed to increase the family of possible supports and A though a kind of support is probably too large: in general, I is not closed under unions.

But, if we add $X = X_{\ell,\bar{b}}$ we have to add all "similar" $X' = X_{\ell,\bar{b}'}$. Recall that necessarily our strategy is to look for a support $A' \in I$ for $X_{\ell,\bar{b}}$. So we like to find $A' \in I$ which is a support of X, that is such that that if $f \in \mathscr{F}, A \cup A' \subseteq \mathrm{Dom}(f)$, then $f \restriction A$ induces a mapping of $X_{i,\bar{b}}$ to some $X_{i,\bar{b}'}$, which when $f \restriction A' = \mathrm{id}_{A'}$, satisfies that $X_{i,\bar{b}'}$ will be equal to $X_{i,\bar{b}}$ thus justifying the statement "A' supports X." How? We use our bound on the size of the computation. So we need a dichotomy: either there is $A' \in I$ as above or the number of sets $X_{i,\bar{b}'}$ defined by $\psi_i(x,\bar{b}')$ varying $\bar{b}'$ is too large!!
On this dichotomy hangs the proof.

However, we do not like to state this as a condition on N_t but rather on M. We do not "know" how $\psi_\ell(x,\bar{b}')$ will act but for any possible A' this induces an equivalence relation on the set of images of A' (for this $\mathscr{F}$ has to be large enough).

Actually, we can ignore the ψ_ℓ's and develop set theory of elements demanding each has a support in I through $\mathscr{F}$. Now we break the proof to definition and claims.

We consider several variants of the logic: the usual variant to make preservation clear, and the case with the cardinality quantifier. We use one $\mathscr{F}$ but we could have used $\langle \mathscr{F}_\ell : \ell \le k' \rangle$; in this case actually, for much of the treatment only $\mathscr{F}_0$ would count. The relevant basic family of partial automorphisms is defined in 2.1. Note that the case with cardinality logic, with a stronger assumption is clearer, if you like to concentrate on it, ignore 2.1(4) and read in Definition 2.3 only part (1), ignore 2.9 but read 2.10, ignore 2.17, 2.20 but read 2.18, ignore 2.22, 2.24 but read 2.23.

2.1 The Main Definition.

1) We say $\mathscr{Y} = (M, I, \mathscr{F})$ is a k-system if

- (A) I is a non empty family of subsets of $|M|$ (the universe of the model M) closed under subsets and each singleton belongs to it
  [hint: intended as the possible supports of elements $N_t[M, \Upsilon]$ and as first approximation to the possible supports of the partial automorphisms of M, where M is the model of course; the intention is that M is a finite model]
  Let $I[m] =: \{\bigcup_{\ell=1}^{m} A_\ell : A_\ell \in I \text{ for } \ell = 1, \ldots, m\}$
- (B) $\mathscr{F}$ is a non empty family of partial automorphisms of M such that $f \in \mathscr{F}$ & $A \subseteq \mathrm{Dom}(f)$ & $A \in I \Rightarrow f"(A) \in I$ (recall $f"(A) = \{f(x) : x \in A \cap \mathrm{Dom}(f)\}$; $\mathscr{F}$ is closed under inverse (i.e. $f \in \mathscr{F}_\ell \Rightarrow f^{-1} \in \mathscr{F}_\ell$) and composition and restriction (hence, together with (D) clearly $B \in I[k] \Rightarrow \mathrm{id}_B \in \mathscr{F}$)
- (C) if $f \in \mathscr{F}$ then $\mathrm{Dom}(f)$ is the union of $\le k$ members of I
- (D) if $f \in \mathscr{F}$ and $A_1, \ldots, A_{k-1}, A_k \in I$ and $\ell \in \{1, \ldots, k-1\} \Rightarrow A_\ell \subseteq \mathrm{Dom}(f)$, then for some $g \in \mathscr{F}$ we have

$$f \restriction \bigcup_{\ell=1}^{k-1} A_\ell \subseteq g$$

$$A_k \subseteq \mathrm{Dom}(g)$$

2) Assume $\mathscr{Y}$ is a k-system and $B \in I[m], m \leq k-2$ and $A \in I$

(α) let $H_{\mathscr{Y}}(B,A) = \{g \in \mathscr{F} : \mathrm{Dom}(g) \supseteq B \cup A$ and $\mathrm{id}_B \subseteq g\}$

(β) $\mathscr{E}_{\mathscr{Y}}(B,A)$ is the family of equivalence relations E on $H_{\mathscr{Y}}(B,A)$ such that:

($*$) if $g_1, g_2, g_3, g_4 \in H_{\mathscr{Y}}(B,A)$ and $f \in \mathscr{F}$ satisfies $\mathrm{id}_B \subseteq f, g\text{"}_1(A) \cup g\text{"}_2(A) \subseteq \mathrm{Dom}(f)$ and $g_3 \supseteq f \circ (g_1 \restriction (B \cup A)), g_4 \supseteq f \circ (g_2 \restriction (B \cup A)$ then $g_1 E g_2 \Leftrightarrow g_3 E g_4$.

(this tells you in particular that only $g \restriction A$ matter for determining g/E)

3)

(α) Let $H_{I,m} = H_{\mathscr{Y}_{,m}}$ be the family of functions h from $[m] = \{1, \ldots, m\}$ to $\mathrm{Seq}_I = \{\bar{a} : \bar{a}$ list with no repetitions some $A \in I$; we can look at $\bar{a}$ as a one-to-one function from $[0, \ell g \bar{a})$ onto $A\}$; of course for $f \in \mathscr{F}$ and $\bar{a}, \bar{b} \in \mathrm{Seq}_I$ the meaning of $f(\bar{a}) = \bar{b}$ is $\ell g(\bar{a}) = \ell g(\bar{b})$ and $f(a_i) = b_i$ for $i < \ell g(\bar{a})$. Let $\mathrm{Seq}_{I,A} = \{\bar{a} \in \mathrm{Seq}_I : \mathrm{Rang}(\bar{a}) = A\}$. For $m = 1$ we may identify $h : [m] \to \mathrm{Seq}_I$ with $h(1)$ so $H_{I,1}$ is identified with Seq_I.

(β) for $h \in H_{I,m}$ and $f \in \mathscr{F}$ such that $\bigcup_{i \in [m]} \mathrm{Rang}(h(i)) \subseteq \mathrm{Dom}(f)$ we define $h' = f * h$ as the following function: $\mathrm{Dom}(h') = [m]$ and $h'(i) = f \circ (h(i))$

(γ) let $2r + s \leq k$; for $B \in I[s]$ and $1 \leq m \leq r$ let $E^0_{I,B,m} = E^0_{\mathscr{Y},B,m}$ be the following 2-place relation on $\{h : h : [m] \to \mathrm{Seq}_I\}$:
$h_1 E^0_{I,B,m} h_2$ iff for some $f \in \mathscr{F}$, $\mathrm{id}_B \subseteq f$ and $h_2 = f * h_1$.
If $B = \emptyset$ we may omit it; similarly $m = 1$

(δ) for $2m + s \leq k$ and $B \in I[s]$ let $\mathscr{E}_{I,m}(B) = \mathscr{E}_{\mathscr{Y}_{,m}}(\emptyset)$ be the family of equivalence relations such that for some $h^* \in H_{I,m}$, E is an equivalence relation on the set $\{h \in H_{I,m} : h E^0_{I,m} h^*\}$ which satisfies:

($*$) if $h_1, h_2, h_3, h_4 \in H_{I,m}$, $f \in \mathscr{F}$, $\mathrm{id}_B \subseteq f$, $h_2 = f * h_1, h_4 = f * h_3$ and $\bigcup \{\mathrm{Rang}(h_1(i)) \cup \mathrm{Rang}(h_1(i)) : i \in [m]\} \subseteq \mathrm{Dom}(f)$,
then $h_1 E h_3 \equiv h_2 E h_4$.
If $B = \emptyset$ we may omit it. If $m = 1$ we may omit it.

4) We say that a k-system $\mathscr{Y} = (M, I, \mathscr{F})$ is $(\mathbf{t}, r)$-dichotomical[2] when

$\boxtimes$ if $1 \leq m \leq r$ and $E \in \mathscr{E}_{I,m}(\emptyset)$, then $(\beta)_1 \vee (\beta)_2$ where

$(\beta)_1$ there is $A \in I$ which satisfies:
if $h_1, h_2 \in H_{I,m}$, $f \in \mathscr{F}$, $\mathrm{id}_A \subseteq f$ and $h_2 = f * h_1$ then $h_1 E h_2$

$(\beta)_2$ the number of E-equivalence classes is $> \mathbf{t}(\|M\|)$.

[2]Note that this is how from "there are not too many" we get "there is a support in I"

If we omit r (and write $\mathbf{t}$-dichotomical) we mean $r = [k/2]$, $k \geq 3$.

Note that in parts (3), (4) without loss of generality h is one-to-one.

* * *

2.2 Remark. 1) However, if we shall deal with $\mathscr{L}^* = \mathscr{L}_{\rm card}$ or $\mathscr{L}^* = L_{{\rm card},\mathbf{T}}$ we naturally have to require that $f \in \mathscr{F}$ preserve more. Whereas the $(\mathbf{t}, r)$-dichotomy is used to show that either we try to add too many sets to some $N_t[M, \Upsilon]$ or we have support, the "counting (k, r)-system" assure us that the lifting of f preserves the counting quantifier, and the medium $(\mathbf{t}, r)$-dichotomy will be used in the maximal successor, see 2.24.

2) It causes no harm real in 2.1(3)$(\gamma), (\delta)$ and similarly later, to restrict ourselves to e.g. $r + s \leq k/100, k > 400$.

2.3 Definition. 1) We say $\mathscr{Y} = (M, I, \mathscr{F})$ is a counting (or super) (k, r)-system if:

$\mathscr{Y}$ is a k-system and

$(*)_1$ Assume that $0 \leq m \leq r$ and for $\ell = 1, 2$ we have $B_\ell \in I[m]$ and $E_\ell \in \mathscr{E}_{\mathscr{Y},1}(B_\ell)$. If $f \in \mathscr{F}, f$ maps B_1 onto B_2 and f maps E_1 to E_2 (see 2.4(1)), then $|{\rm Dom}(E_1)/E_1| = |{\rm Dom}(E_2)/E_2|$.

(This should be good for analyzing the model $N_t[M, \Upsilon, \mathbf{t}]$). If we omit r (write counting k-system) we mean $r = k - 2, k \geq 3$.
2) We say that the k-system $\mathscr{Y} = (M, I, \mathscr{F})$ is medium $(\mathbf{t}, k, r)$-system if

$(*)_2$ Assume that $1 \leq m \leq r$ and for $\ell = 1, 2$ we have $B_\ell \in I[m]$ and $E_\ell \in \mathscr{E}_{\mathscr{Y},1}(B_\ell)$. If $f \in \mathscr{F}, f$ maps B_1 onto B_2 and f maps E_1 to E_2 (see 2.4), then $|{\rm Dom}(E_1)/E_1| = |{\rm Dom}(E_1)/E_2|$ or both are $> \mathbf{t}(\|M\|)$.

3) We omit r if $r = k - 2 \geq 1$ (see 2.8 below).

Note that 2.4 is closed to 2.8 and 2.7(2)-(4).

2.4 Observation. Let $\mathscr{Y} = (M, I, \mathscr{F})$ is a k-system.
1)

(α) if $2m + s \leq k$ and $B \in I[s]$, then $E^0_{\mathscr{Y},B,m}$ is an equivalence relation satisfying $(*)$ of Definition 2.1(3)(δ)

(β) the following two conditions on $B \in I[s], m \leq (k - s)/2, s \leq k$ and $\mathscr{G}$ are equivalent:

- (i) $\mathscr{G}$ is an equivalence class of $E^0_{I,B,m}$
- (ii) $\mathscr{G}$ is the domain of some $E \in \mathscr{E}_{I,m}(B)$

(γ) if $k^* = 2m + s \leq k$ and $\mathscr{F}^* = \{f \restriction A : f \in \mathscr{F}, A \in I[k^*]\}$, and $\mathscr{Y}^* = (M, I, \mathscr{F}^*)$ then $\mathscr{Y}^*$ is a k^*-system and for each $B \in I[s]$ we have $\mathscr{E}_{\mathscr{Y},m}(B) = \mathscr{E}_{\mathscr{Y}^*,m}(B)$ and $E^0_{\mathscr{Y},B,m} = E^0_{\mathscr{Y}^*,B,m}$.

(δ) if $B_1, B_2 \in I[k-2], f \in \mathscr{F}, f$ maps B_1 onto B_2 and $g'_1, g''_1 \in \mathrm{Seq}_I$, $\mathrm{Rang}(g'_1) \cup \mathrm{Rang}(g'_2) \subseteq \mathrm{Dom}(f)$, then $g'_2 =: f \circ g'_1$ belongs to Seq_I and $g''_2 = f \circ g''_1$ belongs to Seq_I and $g'_1 E^0_{I,B_1,1} g''_1 \Leftrightarrow g'_2 E^0_{I,B_2,1} g''_2$

(ε) If $B_1 \in I[k-2], A_2 \in I, f \in \mathscr{F}, B_1 \subseteq \mathrm{Dom}(f), B_2 = f``(B_1)$, then we can define $F^{\mathrm{eq}}_{f,B_1,B_2}(E) \in \mathscr{E}_{I,1}(B_2)$ for $E \in \mathscr{E}_{I,1}(B_1)$ (the image of E by f) by: if $f \restriction B_1 \subseteq g \in \mathscr{F}, \bar{a}_1, \bar{a}_2 \in \mathrm{Seq}_I, g$ maps $\bar{a}_1$ to $\bar{a}^*_1$ and g maps $\bar{a}_2$ to $\bar{a}^*_2$ then $\bar{a}_1 E \bar{a}_2 \Leftrightarrow \bar{a}^*_1 F^{\mathrm{eq}}_{f,B_1,B_2}(E) \bar{a}^*_2$.

2) Let $2r + s \le k$,
(δ), (ε), parallely to part (1) with $m \le r, h_\ell \in H_{I,m}, B_\ell \in I[s]$.

Proof. Straight, e.g.:
Part (1), Clause (α): We use: $\mathscr{F}$ contains id_C wherever $C \in I[k]$ (for reflexivity), $\mathscr{F}$ closed under inverting (for symmetry) and is closed under composition (for transitivity).

$\square_{2.4}$

2.5 Discussion 1) In the system $\mathscr{Y} = (M, I, \mathscr{F})$ we deal only with partial automorphisms of M, we need to lift them to the models $N_t[M, \Upsilon]$ or actually $N^+_t[M, \Upsilon]$ or $N^+_t[M, \Upsilon, \mathbf{t}]$ appearing in Definition 1.3; this motivates the following definition 2.6. (We more generally define liftings to (M, Υ)-candidates).
2) Note that here probably it is more natural if in the definition of k-system $\mathscr{Y}$, we replace the relations "$f \subseteq g$", "$f = g \restriction A$", "$f \supseteq g \restriction A$" on $\mathscr{F}$ by abstract ones (so $\mathscr{F}$ will be an index set). Also in Definition 2.6 we could demand more properties which naturally holds (similarly in Definition 2.13, e.g. if you satisfy the properties of "a set B is $\mathfrak{Z}$-support of x" you are one).

2.6 Definition. 1) Let $\mathscr{Y} = (M, I, \mathscr{F})$ be an k-system, M a τ-model, Υ is an inductive scheme for $\mathscr{L}^*(\tau^+)$ and $\mathbf{m}_1 = \mathbf{m}^\Upsilon_1$.

We say that $\mathfrak{Z} = (N, \bar{P}, G, R) = (N^{\mathfrak{Z}}, \bar{P}^{\mathfrak{Z}}, G^{\mathfrak{Z}}, R^{\mathfrak{Z}})$ is a Υ-lifting or $\mathbf{m}_1$-lifting of $\mathscr{Y}$ if

(a) $(N, \bar{P})$ is an $(M, \mathbf{m}_1)$-candidate so N is a transitive submodel of set theory i.e. of $V_\infty[M]$ with M as its set of urelements and the relations of M (see Definition 1.4(1))

(b) G is a function with domain $\mathscr{F}$

(c) for $f \in \mathscr{F}$
- (α) $G(f)$ is a function with domain $\subseteq N, f \subseteq G(f)$, moreover $f = G(f) \restriction M$ and
- (β) $G(f)$ is a partial automorphism of N

(d) if $f \in \mathscr{F}, g \in \mathscr{F}, f \subseteq g$ then $G(f) \subseteq G(g)$

(e) R is a two-place relation written xRy such that
$xRy \Rightarrow x \in I \ \& \ y \in N$
[we say: x is a $\mathfrak{Z}$-support of y]

(f)

(α) if ARy and $f \in \mathscr{F}, A \subseteq \mathrm{Dom}(f)$, then $y \in \mathrm{Dom}(G(f))$ and $f \restriction A = \mathrm{id}_A \Rightarrow G(f)(y) = y$

(β) if $f \in \mathscr{F}$ and $y \in \mathrm{Dom}(G(f))$ (hence $y \in N$) then some a $\mathfrak{Z}$-support of y is included in $\mathrm{Dom}(f)$

(g) $(\forall y \in N)(\exists A \in I)ARy$
[i.e. every element of N has a $\mathfrak{Z}$-support]

(h) if $A \in I$ and $A \subseteq \mathrm{Dom}(f), y \in \mathrm{Dom}(G(f))$ then $ARy \Leftrightarrow f"(A)R(G(f)(y))$

(i) for $f \in \mathscr{F}$ we have $G(f^{-1}) = (G(f))^{-1}$

(j) for $f_1, f_2 \in \mathscr{F}, f = f_2 \circ f_1$ we have[3] $G(f) \subseteq G(f_2) \circ G(f_1)$

(k) if $\bar{a} \in {}^{\mathbf{m}_1(\ell)}(\mathrm{Dom}(f))$ and $f \in \mathscr{F}$ and $f(\bar{a})$ is well defined, then $\bar{a} \in P^{\mathfrak{Z}}_\ell \equiv f(\bar{a}) \in P^{\mathfrak{Z}}_\ell$; moreover $\emptyset R c_\ell$ if $P^{\mathfrak{Z}}_\ell$ is the individual constant $c_\ell = c^{\mathfrak{Z}}_\ell$ when c_ℓ is well defined (see Definition 1.1(2)(F); this implies that $G(f)$ is a partial automorphism of $(N, \bar{P})$).

We may write $\mathbf{m}_1 = \mathbf{m}^{\mathfrak{Z}}_1$, recall that $\mathbf{m}_1(k)$ gives the arity of P_k and the information is it a relation or (possibly partial) function.

2.7 Fact: Let $\mathscr{Y} = (M, I, \mathscr{F})$ be a k-system, and $\mathfrak{Z}$ be an $\mathbf{m}_1$-lifting of $\mathscr{Y}$.
1) The $0 - \Upsilon$-lifting (in Definition 2.12) exists and is a lifting (see Definition 2.6).
2) If $f_1, f_2 \in \mathscr{F}$ and A is a $\mathfrak{Z}$-support of $y \in N$ then

$$f_1 \restriction A = f_2 \restriction A \ \ \& \ \ A \subseteq \mathrm{Dom}(f_1) \quad \Rightarrow \quad f_1(y) = f_2(y).$$

3) From $\mathfrak{Z}$ we can reconstruct $\mathscr{Y}, \mathscr{F}, \mathbf{m}_1$; and if $I = \{A :$ for some $B, A \subseteq B$ and B is a $\mathfrak{Z}$-support of some $y \in N\}$ then we can reconstruct also I (so the whole $\mathscr{Y}$).

Proof. 1) Easy [compare with 2.4, 2.8].
2) Let $y' = G(f_2)(y)$, let $A_1 = A, A_2 = f"_2(A_1)$ hence as A is a $\mathfrak{Z}$-support of y and $A \subseteq \mathrm{Dom}(f_1)$ necessarily y' is well defined and $A_2 R^{\mathfrak{Z}} y'$ (see Definition 2.6(1) clause (h)). We know that f_2^{-1} and $f_2^{-1} \circ f_1$ belongs to $\mathscr{F}$ (see Definition 2.6(1) clauses (i) and (j)). We also know that $A_2 \subseteq \mathrm{Dom}(f_2^{-1})$ so as $A_2 R^{\mathfrak{Z}} y'$ (see above) we have $y' \in \mathrm{Dom}(G(f_2^{-1}))$ (see Definition 2.6(1), clause $(f)(\alpha)$) and $(G(f_2^{-1}))(y') = y$ (by Definition 2.6(1), clause (i) as $(G(f_2))(y) = y'$ by the choice of y').
Clearly $A = A_1 \subseteq \mathrm{Dom}(f_2^{-1} \circ f_1)$ hence (see Definition 2.6(1), clause $(f)(\beta)$) we have $y \in \mathrm{Dom}(G(f_2^{-1} \circ f))$.

[3] So maybe x even has support A_ℓ, $\mathrm{id}_{A_\ell} \subseteq f_\ell$ for $\ell = 1, 2$ but x has no support $\subseteq \mathrm{Dom}(f)$

But $\mathrm{id}_A \subseteq f_2^{-1} \circ f_1$, so as $AR^{\mathfrak{Z}} y$ we have $y = (G(f_2^{-1} \circ f))(y)$. By Definition 2.6(1), clause (j) we have, as the left side is well defined:

$$(G(f_2^{-1} \circ f_1))(y) = ((G(f_2^{-1})) \circ (G(f_1)))(y)$$

and trivially

$$(G(f_2^{-1})) \circ (G(f_1)))(y) = (G(f_2^{-1}))((G(f_1))(y)).$$

By the last three equations $y = (G(f_2^{-1} \circ f_1))(y) = (G(f_2^{-1}))((G(f_1))(y))$, but by the above we note $y = (G(f_2^{-1}))(y')$. So, as $G(f_2^{-1})$ is one-to-one (having an inverse), we have $(G(f_1))(y) = y'$, now as y' was defined as $(G(f_2))(y)$ we are done.

3) Straightforward. $\square_{2.7}$

Note that 2.8 is close to 2.4 and 2.12(2)-(4).

2.8 Definition/Claim. Let $\mathscr{Y} = (M, I, \mathscr{F})$ be a k-system and $\mathfrak{Z} = (N, \bar{P}, G, R)$ be an $\mathbf{m}_1$-lifting of $\mathscr{Y}$.

1) For $B \subseteq N$ let $\mathbf{E}_B = \mathbf{E}_B^{\mathscr{Y},\mathfrak{Z}}$ be the following 2-place relation on N:

$$x_1 \mathbf{E}_B x_2 \ \underline{\text{iff}} \ \text{for some } f \in \mathscr{F} \text{ we have } \mathrm{id}_B \subseteq f \text{ and } (G(f))(x_1) = x_2.$$

2) If $B \in I[k-2]$ so $k \geq 3$ then

(α) $\mathbf{E}_B$ is an equivalence relation on N

(β) if $f \in \mathscr{F}, B \subseteq \mathrm{Dom}(f)$ then f maps $\mathbf{E}_B$ to $\mathbf{E}_{f''(B)}$ which means:

$$f \restriction B \subseteq g \in \mathscr{F} \ \& \ \bigwedge_{\ell<2} (G(g))(x_\ell) = y_\ell \Rightarrow [x_1 \mathbf{E}_B x_2 \equiv y_1 \mathbf{E}_{f''(B)} y_2]$$

(γ) if $B \subseteq \mathrm{Dom}(f)$ then there is a one-to-one function $F = F_{f,B}$ from $N/\mathbf{E}_B$ onto $N/\mathbf{E}_{f''(B)}$ such that:

$(*)_1$ for $x_1, x_2 \in N$ we have: $(\exists g)(f \restriction B \subseteq g \in \mathscr{F} \ \& \ G(g)(x_1) = x_2) \Leftrightarrow F(x_1/\mathbf{E}_B) = x_2/\mathbf{E}_{f''(B)}$

(δ) if $x \in N$ and $AR^{\mathfrak{Z}} x$ and $\bar{a} \in \mathrm{Seq}_{I,A}$ then there is an equivalence relation $E \in \mathscr{E}_{\mathscr{Y}_1}(B)$ with domain $\{f(\bar{a}) : f \text{ extend } \mathrm{id}_B \text{ and } A \subseteq \mathrm{Dom}(f)\}$ such that:

$(*)_2$ if $f_\ell \in \mathscr{F}$, $\mathrm{id}_B \subseteq f_\ell$ and $A \subseteq \mathrm{Dom}(f_\ell)$ for $\ell = 1, 2$ then $G^{\mathfrak{Z}}(f_1)(x) = G^{\mathfrak{Z}}(f_2)(x) \Leftrightarrow f_1(\bar{a}) E f_2(\bar{a})$

$(*)_3$ $|x/\mathbf{E}_B| = |\mathrm{Dom}(E)/E|$

(ε) if f, F are as in clause (γ) and $x_1, x_2 \in N$ then

$(*)_4$ if $F(x_1/\mathbf{E}_B) = x_2/\mathbf{E}_B$ and $\mathscr{Y}$ is a counting k-system, then $|x_1/\mathbf{E}_B| = |x_2/\mathbf{E}_B|$

$(*)_5$ if $F(x_1/\mathbf{E}_B) = x_2/\mathbf{E}_B$ and $\mathscr{Y}$ is a medium $(\mathbf{t}, k)$-system, then $|x_1/\mathbf{E}_B| = |x_2/\mathbf{E}_B|$ or both are $> \mathbf{t}(M)$.

Proof of (2).

Clause (α):

Reflexivity:
For $x \in N$ choose a $\mathfrak{Z}$-support $A \in I$, so we can find $f \in \mathscr{F}$ extending $\mathrm{id}_{B \cup A}$ hence $G(f)$ maps x to itself, hence $x \mathbf{E}_B x$.

Symmetry:
If $f \in \mathscr{F}$ witnesses $x \mathbf{E}_B y$ then $f^{-1} \in \mathscr{F}$ witnesses $y \mathbf{E}_B x$.

Transitivity:
If $x_0 \mathbf{E}_B x_1, x_1 \mathbf{E}_B x_2$ let f_1 witness $x_0 \mathbf{E}_B x_2$ and let f_2 witness $x_1 \mathbf{E}_B x_2$, now let $A_0 \subseteq \mathrm{Dom}(f_1)$ be a $\mathfrak{Z}$-support of x_0, so $A_1 = f''_1(A_0) \subseteq \mathrm{Rang}(f_1)$ is a $\mathfrak{Z}$-witness of x_1, now let $A_1^* \subseteq \mathrm{Dom}(f_2)$ be a $\mathfrak{Z}$-support of X_1, so $B \cup A_1^* \in I[k-1]$ hence without loss of generality $A_1 \subseteq \mathrm{Dom}(f_2)$ hence $A_2 = f''_2(A_1)$ is a $\mathfrak{Z}$-support of x_2, so $x_1 \in \mathrm{Dom}(G(f_2 \circ f_1))$ and $G(f_2 \circ f_1) \subseteq G(f_1) \circ G(f_2)$ hence $G(f_2 \circ f_1)(x_1) = x_2$ as required.

Clause (β):
So assume $f \restriction B \subseteq g \in \mathscr{F}$ and $(G(g))(x_\ell) = y_\ell$, for $\ell = 1, 2$ and we should prove $x_1 \mathbf{E}_B x_2 \equiv y_1 \mathbf{E}_{f''(B)} y_2$. It suffices to prove $x_1 \mathbf{E}_B x_2 \Rightarrow y_1 \mathbf{E}_{f''(B)} y_2$ (as applying it to $B' = f''(B), f' = f^{-1}, g' = g^{-1}, y_1, y_2, x_1, x_2$ we get the other implication). As $x_1 \mathbf{E}_B x_2$ we can find a witness h, i.e., $\mathrm{id}_B \subseteq h \in \mathscr{F}$ and $(G(h))(x_1) = x_2$. Let $A_1 \subseteq \mathrm{Dom}(h)$ be a $\mathfrak{Z}$-support of x_1 and let $A_2 = h''(A_1)$, so A_2 is a $\mathfrak{Z}$-support of x_2.

If $B \in I[k-4]$ without loss of generality $A_1, A_2 \subseteq \mathrm{Dom}(g)$, and let $A_1^* = f''(A_1)$, $A_2^* = f''(A_2)$, lastly let $g^* = g \circ h \circ g^{-1}$. Now $(G(g^*))(y_1) = y_2$, $\mathrm{id}_{f''(B)} \subseteq g^*$ so g^* witnesses $y_1 \mathbf{E}_{f''(B)} y_2$ as required.

But maybe $B \notin I[k-4]$, still $B \in I[k-2]$; now for $\ell = 1, 2$, as $(G(f))(x_\ell) = y_\ell$ there is a $\mathfrak{Z}$-support C_ℓ of x_ℓ such that $C_\ell \subseteq \mathrm{Dom}(g)$ and let $C'_\ell = g''(C_\ell)$. So we can find, for $\ell = 1, 2$ a function $g_1 \in \mathscr{F}$ such that $g \restriction (B \cup C_1) \subseteq g_1$ and $A_1 \subseteq \mathrm{Dom}(g_2)$ and also there is $h_1 \in \mathscr{F}$ such that $h \restriction (B \cup A_1) \subseteq h_1$ and $C_1 \subseteq \mathrm{Dom}(h_1)$. So $h_1 \circ g_1^{-1}$ extends $(g \restriction B)^{-1}$ and $C'_1 \subseteq \mathrm{Dom}(g_1^{-1})$, $(g_1^{-1})''(C'_1) = C_1 \subseteq \mathrm{Dom}(h_1)$ but C'_1 is a $\mathfrak{Z}$-support of y_1. Hence $(G(h \circ g_1^{-1}))(y_1)$ is well defined and equal to $(G(h) \circ G(g_1^{-1}))(y_1) = (G(h))((G(g_1^{-1}))(y_1)) = (G(h_1))(x_1) = x_2$. Let $g_2 \in \mathscr{F}$ be such that $g \restriction (B \cup A_2) \subseteq g_2$ and $(h_1 \circ g_1^{-1})''(C'_1) \subseteq \mathrm{Dom}(g_2)$ and similarly we get $g_2^{-1} \circ h_1 \circ g_1^{-1}$ extends $\mathrm{id}_{g''(B)}$ and $(G(g_2^{-1} \circ h_1 \circ g_1^{-1}))(y_1) = y_2$, so we are done.

Clause (γ), (δ), (ε):
Should be clear. $\square_{2.8}$

Quite naturally for such a $\mathbf{m}_1$-lifting $\mathfrak{Z}$ of $\mathscr{Y}$ the family $\{G(f) : f \in \mathscr{F}^{\mathscr{Y}}\}$ helps us to understand first order logic on $(N^{\mathfrak{Z}}, \bar{P}^{\mathfrak{Z}})$.

2.9 Claim. *Assume $\mathscr{Y} = (M, I, \mathscr{F})$ is a k-system and $\mathfrak{Z} = (N, \bar{P}, G, R)$ is an $\mathbf{m}_1$-lifting of $\mathscr{Y}$.*
Then

(∗) *Assume $\varphi(\bar{x})$ is first order and $k \geq$ quantifier depth$(\varphi(\bar{x})) + \ell g(\bar{x})$, or just $\varphi(\bar{x}) \in \mathscr{L}_{\infty,k}$ which means:*
every subformula of $\varphi(\bar{x})$, (e.g. φ itself) has $\leq k$ free variables.

If $\bar{a} \in {}^{\ell g(\bar{x})}N, A_\ell R a_\ell$ and $A_\ell \subseteq \mathrm{Dom}(f)$ (hence $a_\ell \in$ Dom$(G(f))$) for $\ell < \ell g(\bar{x})$ and $f \in \mathscr{F}$ then

$$(N, \bar{P}) \models \text{“}\varphi[\ldots, a_\ell, \ldots]_{\ell < \ell g(\bar{x})}\text{”} \Leftrightarrow (N, \bar{P}) \models \text{“}\varphi[\ldots, G(f)(a_\ell), \ldots]_{\ell < \ell g(\bar{x})}\text{”}$$

Proof. We prove this by induction on the quantifier depth of φ. Let $m = \ell g(\bar{x})$ so $\bar{x} = \langle x_\ell : \ell < m \rangle$ and without loss of generality $m \leq k$.

Case 1: φ atomic.
As $G(f)$ is a partial automorphism of N and even $(N, \bar{P})$ this should be clear.

Case 2: $\varphi = \neg\psi$ or $\varphi = \psi_1 \wedge \psi_2$ or $\varphi = \varphi_1 \vee \varphi_2$.
Straight.

Case 3: $\varphi = \varphi(\bar{x}) = (\exists y)\psi(y, \bar{x})$.
Without loss of generality y is not a dummy variable in ψ, that is has a free occurrence. Let $\bar{x} = \langle x_0, \ldots, x_{m-1} \rangle$ so $m < k$.
As $G(f^{-1}) = G(f)^{-1}$ it is enough to prove $N \models \text{“}\varphi[a_0, \ldots, a_{m-1}]\text{”} \Rightarrow N \models \text{“}\varphi[G(f)(a_0), \ldots, G(f)(a_{m-1})]\text{”}$.
So we assume the left side, i.e.

$(*)_1$ $N \models \varphi[a_0, \ldots, a_{m-1}]$,
hence for some $a^* \in N$ we have

$(*)_2$ $N \models \psi[a^*, a_0, \ldots, a_{m-1}]$.

Necessarily a^* has a $\mathfrak{Z}$-support $A^* \in I$.
Now $k \geq m + 1$ and $\mathscr{Y}$ is a k-system hence there is $f^* \in \mathscr{F}$ such that $f \restriction (\bigcup_{\ell<m} A_\ell) \subseteq f^*$ and $A^* \subseteq$ Dom(f^*). So each of $a^*, a_0, \ldots, a_{m-1}$ has a $\mathfrak{Z}$-support included in Dom(f^*) hence by the induction hypothesis applied to $\psi[a^*, a_0, \ldots, a_{m_1}]$ and $(*)_2$ we have

$(*)_3$ $N \models \psi[G(f^*)(a^*), G(f^*)(a_0), \ldots, G(f^*)(a_{m-1})]$.

So by the definition of $\models$ we get

$(*)_4$ $N \models (\exists y)\psi(y, G(f^*)(a_0), \ldots, G(f^*)(a_{m-1}))$.

But for $\ell < m$, the set A_ℓ is a $\mathfrak{Z}$-support of a_ℓ and $f^* \restriction A_\ell = f \restriction A_\ell$ hence $(G(f^*))(a_\ell) = G(f)(a_\ell)$ so

$(*)_5$ $N \models (\exists y)\psi(y, G(f)(a_0), \ldots, G(f)(a_{m-1}))]$.

But $\varphi(\bar{x}) = (\exists y)\psi(y, \bar{x})$ so we are done. $\square_{2.9}$

Having dealt with first order logic we should deal with cardinality logic (actually any of the variants we mention). Here we use the counting version, really naturally the medium version suffices but for it we have to use more "bookkeeping" of the various things used, and the reader can use only this smoother case.
Note that if we like to add cardinality quantifiers on pairs we need $s \geq 2$, etc., but we may create the set of pairs in N_t so not so necessary.

2.10 Claim. *Assume that $\mathscr{Y}$ is a counting k-system (see Def 2.3(1)) and $\mathfrak{Z} = (N, \bar{P}, G, R)$ is an $\mathbf{m}_1$-lifting of $\mathscr{Y}$.*
Then

- $(*)$ *assume $\varphi(\bar{x}) \in \mathscr{L}_{\mathrm{card},\mathbf{T}}$ or $\varphi(\bar{x}) \in \mathscr{L}_{\mathrm{card}}$ (can have both kinds quantifiers; recall that $\mathscr{L}_{\mathrm{f.o.\ +\ na}}$ is included in a special case of $\mathscr{L}_{\mathrm{card},\mathbf{T}}$) and every subformula of $\varphi(\bar{x})$, including $\varphi(\bar{x})$ itself, has $< k$ free variables and $m = \ell g(\bar{x}) < k$*
- $(\alpha)_{\varphi(\bar{x})}$ *if $f \in \mathscr{F}$ and A_ℓ is a $\mathfrak{Z}$-support of a_ℓ and $A_\ell \subseteq$ Dom(f) for $\ell < m$ then:*

$$(N, \bar{P}) \models \text{“}\varphi[a_0, \ldots, a_{m-1}]\text{”} \quad \textit{iff} \quad (N, \bar{P}) \models \text{“}\varphi[G(f)(a_0), \ldots, G(f)(a_{m-1})]\text{”}$$

- $(\beta)_{\varphi(\bar{x})}$ *if $f \in \mathscr{F}$ and A_ℓ is a $\mathfrak{Z}$-support of a_ℓ for $\ell = 1, \ldots, m-1$ and $A_\ell \subseteq$ Dom(f) then the sets $\{b \in N : (N, \bar{P}) \models \text{“}\varphi[b, a_1, \ldots, a_{m-1}]\text{”}\}$, and $\{b \in N : (N, \bar{P}) \models \text{“}\varphi[b, G(f)(a_1), \ldots, G(f)(a_{m-1})]\text{”}\}$ have the same number of elements.*

Proof. We prove by induction on the quantifier depth of φ.
We first show that

$$\boxtimes \ (\alpha)_{\varphi(\bar{x})} \Rightarrow (\beta)_{\varphi(\bar{x})}$$

Why? So assume $(\alpha)_{\varphi(\bar{x})}$, and $a_1, \ldots, a_{m-1} \in N$ be given (where $\ell g(\bar{x}) = m$) and also $f \in \mathscr{F}$ and $A_1, \ldots, A_{m-1} \in N, A_\ell$ is a $\mathfrak{Z}$-support of a_ℓ for $\ell = 1, \ldots, m-1$ such that $A_\ell \subseteq$ Dom(f), and we should prove the equality in $(\beta)_{\varphi(\bar{x})}$. Let a^i_ℓ be a_ℓ if $i = 1$ and $(G(f))(a_\ell)$ if $i = 2$. Let A^i_ℓ be A_ℓ if $i = 1$ and $(G(f))''(A_\ell)$ if $\ell = 2$.
Let $B_i = \bigcup_{\ell=1}^{m-1} A^i_\ell$ so $B_i \in I[m-1]$ and f maps B_1 onto B_2.

By Definition 2.8(1) we know that $\mathbf{E}_{B_\ell}$ is an equivalence relation on N and, see Definition 2.8(2), clause $(\gamma), (\delta)$, the function $F = F_{f,B_1}$ satisfies

- (i) F is a one-to-one function from $N/\mathbf{E}_{B_1}$ onto $N/\mathbf{E}_{B_2}$
- (ii) $f \restriction B_1 \subseteq g \in \mathscr{F}$ & $(G(g))(x_1) = x_2 \Rightarrow (F(x_1)/\mathbf{E}_{B_1}) = x/\mathbf{E}_{B_2}$

(iii) for every $x_1 \in N, A_1 \in I$ such that $A_1 R^{\mathfrak{Z}} x_1$, and $\bar{a}_1 \in \mathrm{Seq}_{I,A_1}$ for some $E = E_{x_1} \in \mathscr{E}_{\mathscr{Y},1}(B_1)$ we have $\mathrm{Dom}(E) = x_1/\mathbf{E}_{B_1}$ and

($*$) if $[\mathrm{id}_{B_1} \subseteq f_\ell \ \&\ A_1 \subseteq \mathrm{Dom}(f_\ell)]$ for $\ell = 1,2$ then $f_1(\bar{a}_1) E_x f_2(\bar{a}_2) \Leftrightarrow (G(f_1))(x_1) = (G(f_2))(x_1)$

(iv) if $x_1 \in N, x_2 \in N, (G(g))(x_1) = x_2$ for some $g \in \mathscr{F}$ such that $f \restriction B_1 \subseteq g$, then $|(x_1/\mathbf{E}_{B_1})/E_{x_1}| = |(x_2/\mathbf{E}_{B_2}/E_{x_2}|$.

Hence it suffices to prove, assuming $F(x_1/\mathbf{E}_{A_1}) = x_2/\mathbf{E}_{A_2}$ that

$$N \models \text{“}\varphi[x_1, a_1, \ldots, a_{m-1}]\text{”} \Leftrightarrow N \models \text{“}\varphi[x_2, G(f)(a_1), \ldots, G(f)(a_{m-1})]\text{”}.$$

As we can replace f by a suitable extension of $f \restriction B_1$, without loss of generality there are $y_1 \in x_1/\mathbf{E}_{B_1}$ and $y_2 \in x_2/\mathbf{E}_{B_2}$ such that $(G(f))(y_1) = y_2$. We can find $C_1 \subseteq \mathrm{Dom}(f)$ which is a $\mathfrak{Z}$-support of y_1.

As $x_1 \mathbf{E}_{B_1} y_1$ we can find $f_1 \in \mathscr{F}$ such that $\mathrm{id}_{B_1} \subseteq f_1$ and $(G(f_1))(x_1) = y_1$; as $m < k$, without lose of generality $C_1 \subseteq \mathrm{Rang}(f_1)$ and let $C_0 = (f_1^{-1})\text{”}(C_1)$, let $C_2 = f\text{”}(C_1)$. As $x_2 \mathbf{E}_{B_1} y_2$ we can find $f_2 \in \mathscr{F}$ such that $\mathrm{id}_{B_2} \subseteq f_2$ and $(G(f_2))(y_2) = x_2$; as $m < k$ without lose of generality $C_2 \subseteq \mathrm{Dom}(f_2)$ and let $C_3 = f\text{”}_2(C_2)$. So $f' = f_2 \circ f \circ f_1$ belongs to $\mathscr{F}$ and extends $f \restriction B_1$ and it maps C_0 onto C_3 hence $y_1 \in \mathrm{Dom}(G(f'))$, and clearly $(G(f_1))(x_1) = y_1, (G(f))(y_1) = y_2, (G(f_2))(y_2) = x_2$ hence $(G(f'))(x_1) = x_2$ and, of course, $B_\ell \subseteq \mathrm{Dom}(f')$ hence applying $(\alpha)_{\varphi(\bar{x})}$ to $f', x_1, a_1, \ldots, a_{m-1}$ we get

$$(N, \bar{P}) \models \text{“}\varphi[x_1, a_1, \ldots, a_{m-1}]\text{”} \Leftrightarrow (N, \bar{P}) \models \text{“}\varphi[x_2, G(f)(a_1), \ldots, G(f)(a_{m-1})]\text{”}$$

recalling $x_2 = (G(f'))(x_1)$. So $\boxtimes$ holds.

Now the inductive proof of $(\alpha)_{\bar{\varphi}}$ is separated to cases. The case φ atomic, $\varphi = \neg\varphi, \varphi = \psi_1 \wedge \psi_2, \varphi = (\exists y)(\psi(y, \bar{x}))$ works as in the proof of 2.9. The new cases hold because $(\beta)_\psi$ hold by the induction hypothesis + $\boxtimes$. $\square_{2.10}$

2.11 Claim. *We can weaken in 2.10 the assumption on $\mathscr{Y}$ to “medium $(\mathbf{t}, k)$-dichotomical” provided that:*

$\boxtimes$ *if $B \in I[m], 1 \le m \le r$, then every equivalence class of $\mathbf{E}_B^{\mathscr{Y},\mathfrak{Z}}$ (so a subset of $N^{\mathfrak{Z}}$, see Definition 2.8(1)) has $\le \mathbf{t}(M^{\mathscr{Y}})$ members.*

Proof. Straightforward.

2.12 Definition. For $\mathscr{Y} = (M, I, \mathscr{F})$ a k-system, the $0 - \Upsilon$-lifting or the $0 - \mathbf{m}_1$-lifting if $\mathbf{m}_1 = \mathbf{m}_1^{\Upsilon}$ is $(M, \bar{P}, G, R)$ where

(a) G is the identity on $\mathscr{F}$

(b) $ARy \Leftrightarrow A \in I \ \&\ y \in A$

(c) each P_ℓ is the empty relation.

Clearly our intention requires us for a k-system $\mathscr{Y}$, to move from an Υ-lifting $\mathfrak{Z}_t = (N_t[M, \Upsilon, \mathbf{t}], \bar{P}_t[M, \Upsilon, t], G_t, R_t)$ to a Υ-lifting

$$\mathfrak{Z}_{t+1} = (N_{t+1}[M, \Upsilon], \bar{P}_{t+1}[M, \Upsilon], G_{t+1}, R_{t+1}).$$

Toward this aim naturally in Definition 2.13 below we define for Υ-lifting $\mathfrak{Z}$ some successors, and in Claim 2.16 we prove what they satisfy.
In Definition 2.13(6) we can define "$\mathfrak{Z}'$ is the full $\mathbf{t}$-successor of $\mathfrak{Z}$" (both $\mathbf{m}_1$-liftings of a k-system $\mathscr{Y}$).

2.13 Definition. Let $\mathscr{Y} = (M, I, \mathscr{F})$ be a k-system and $\mathfrak{Z} = (N, \bar{P}, G, R)$ be an $\mathbf{m}_1$-lifting of $\mathscr{Y}$.
1) We say X is good or $(\mathscr{Y}, \mathfrak{Z})$-good if

- (a) X a subset of N
- (b) for some $A \in I$ we have "A supports X" (for our $\mathscr{Y}$ and $\mathfrak{Z}$) which means: if $f \in \mathscr{F}, BRy$ (so $B \in I, y \in N$) and $A \cup B \subseteq \mathrm{Dom}(f)$, and $f \restriction A = \mathrm{id}_A$ then $y \in X \Leftrightarrow (G(f))(y) \in X$
  (note: $(G(f))(y)$ is well defined by clause (f) of 2.6)
- (c) $X \notin N$.

2) Let $\mathscr{P} = \mathscr{P}_{\mathscr{Y},\mathfrak{Z}}$ be the family of good subsets of N, let $\mathscr{R} = \mathscr{R}_{\mathscr{Y},\mathfrak{Z}}$ be the two-place relation defined by: $A \mathscr{R} X$ iff A supports X, i.e. (b) of part (1) holds.
3) For $f \in \mathscr{F}$ we define a function $G^+(f) = G^+_{\mathscr{Y},\mathfrak{Z}}(f)$ with domain $\subseteq \mathscr{P}_{\mathscr{Y},\mathfrak{Z}} \cup N$ as follows, (well, now $(G^+(f))(X_1) = X_2$ is just a relation, but by 2.14(1) clause (ii) below it is a function)

- (α) For good X such that $A \mathscr{R} X, A \in I$ when $A \subseteq \mathrm{Dom}(f)$ we let $(G^+(f))(X) = \{y \in N :$ for some $g \in \mathscr{F}$ and $y' \in X$ we have $f \restriction A \subseteq g$ and $G(g)(y') = y\}$
- (β) $G^+(f) \restriction N = G(f)$.
  Note that no contradition arises between clauses (α) and (β) because of clause (c) in part (1).

4) We define $E = E_{\mathscr{Y},\mathfrak{Z}}$ as the following two place relation: $X_1 E X_2$ iff X_1, X_2 are good subsets of $N^{\mathfrak{Z}}$ and for some $f \in \mathscr{F}$ we have $(G^+(f))(X_1) = X_2$; this is an equivalence relation (see 2.15(2) below).
5) $\mathfrak{Z}' = (N', \bar{P}', G', R')$ is a successor of $\mathfrak{Z}$ if:

- (a) $N \subseteq N' \subseteq N \cup \mathscr{P}_{\mathscr{Y},\mathscr{Z}}$
- (b) $X_1 E_{\mathscr{Y},\mathfrak{Z}} X_2 \ \& \ X_1 \in \mathscr{P}_{\mathscr{Y},\mathfrak{Z}} \ \& \ X_2 \in \mathscr{P}_{\mathscr{Y},\mathfrak{Z}} \Rightarrow [X_1 \in N' \leftrightarrow X_2 \in N']$
- (c) G' is a function with domain $\mathscr{F}$ and for $f \in \mathscr{F}$ the function $G'(f)$ is defined as $G^+(f)$ from part (3) restricted to N'
- (d) R' is $R \cup [\mathscr{R} \restriction (I \times N')]$
- (e) the pair $(N', \bar{P}')$ is an $(M, \mathbf{m}_1)$-candidate; so P'_ℓ is an $\mathbf{m}_1(\ell)$-ary relation or function as dictated by $\mathbf{m}_1$

6) We say $\mathfrak{Z}'$ is a $\bar{\varphi}$-reasonable successor of $\mathfrak{Z}$ if it is a successor and

$(e)'$ $P'_\ell = \{\bar{b} \in N : (N, \bar{P}) \models \varphi_\ell(\bar{b})\}$ but when Υ is i.c. (see Definition 1.1(2), clause (F)) this is so only if $P'_\ell \in N'$ and $P'_\ell = \emptyset$ otherwise (for each $\ell < \mathbf{m}_1$ (so we demand $P'_\ell \in N'$).

We may omit $\bar{\varphi}$ if clear from the content.
7) We say that $\mathfrak{Z}' = (N', \bar{P}', G', R')$ is a full successor of $\mathfrak{Z}$ if it is a successor of $\mathfrak{Z}$ and $N' = N \cup \mathscr{P}_{\mathscr{Y}_{,\mathfrak{Z}}}$. We say it is the full successor if in addition $P'_\ell = \emptyset$ and it is the full reasonable successor (or reasonable full successor) if it is a full successor which is a reasonable successor.
8) $\mathfrak{Z}' = (N', \bar{P}', G', R')$ is a full $\mathbf{t}$-successor of $\mathfrak{Z}$ if it is a successor of $\mathfrak{Z}$ and

$(a)^*_8$ $N' = N \cup \{X \in \mathscr{P}_{\mathscr{Y}_{,\mathfrak{Z}}} : |X/E_{\mathscr{Y}_{,\mathfrak{Z}}}| \leq \mathbf{t}(\|M\|)\}$.

So if we omit $\mathbf{t}$ we mean $\mathbf{t}(\|N\|) = \infty$.
9) $\mathfrak{Z}' = (N', \bar{P}', G', R')$ is the true $(\Upsilon, \mathbf{t})$-successor of $\mathfrak{Z}$ if $\mathbf{m}_1 = \mathbf{m}_1^\Upsilon$ and $\mathfrak{Z}'$ is a reasonable successor of $\mathfrak{Z}$ and:

$(a)^*_9$ $(N', \bar{P}')$ is the $(\Upsilon, \mathbf{t})$-successor of $(N, \bar{P})$, (see Definition 1.8(2)).

10) $\mathfrak{Z}' = (N', \bar{P}', G', R')$ is the true Υ-successor of $\mathfrak{Z}$ if it is a $\bar{\varphi}^\Upsilon$-reasonable successor of $\mathfrak{Z}$ and:

$(a)^*_{10}$ $(N', \bar{P}')$ is the Υ-successor of $(N, \bar{P})$, (see Definition 1.8(2A)) [this just means the true (Υ, ∞)-successor of $\mathfrak{Z}$].

Note that the names above indicate our intentions, but we have to prove that "$\mathfrak{Z}'$ is a successor of $\mathfrak{Z}$" implies that "$\mathfrak{Z}$ is an $\mathbf{m}_1$-lifting of $\mathscr{Y}$" (done in 2.16), the true $(\Upsilon, \mathbf{t})$-successor of $\mathfrak{Z}$ is a $\mathbf{t}$-successor of $\mathfrak{Z}$ (done in 2.17, 2.18, 2.20) and similarly without the $\mathbf{t}$.

2.14 Claim. *Assume $\mathscr{Y}$ is a k-system, Υ an inductive scheme (so τ is common) and $\mathfrak{Z}$ is an Υ-lifting of $\mathscr{Y}$.*
1) In Definition 2.13(3), if $k \geq 3$ then for $f \in \mathscr{F}$ and $(\mathscr{Y}, \mathfrak{Z})$-good X we have:
(i) if the relation $X_2 = (G^+(f))(X_1)$ holds and $(G(f))(x_1) = x_2$ then $x_1 \in X_1 \equiv x_2 \in X_2$
(ii) the value $(G^+(f))(X)$ does not depend on A, so $G^+(f)$ is well defined.
(iii) if the relation $X_2 = (G^+(f))(X_1)$ holds then X_2 is a $(\mathscr{Y}, \mathfrak{Z})$-good
2) There is a unique object $\mathfrak{Z}'$ which is the full successor of $\mathfrak{Z}$. ; there is a unique object $\mathfrak{Z}'$ which is the reasonable full successor of $\mathfrak{Z}$ and there is a unique object which is the reasonable $\mathbf{t}$-full successor of $\mathfrak{Z}$.
3) If the $\mathfrak{Z}'$ is the true $(\Upsilon, \mathbf{t})$-successor of $\mathfrak{Z}$, then $\mathfrak{Z}'$ is a reasonable successor of $\mathfrak{Z}$ which implies $\mathfrak{Z}'$ is a successor of $\mathfrak{Z}$.
4) There is at most one true successor $\mathfrak{Z}'$ of $\mathfrak{Z}$.

Proof. Easy, using 2.16(2) below for part (2); e.g.
1) Clause (i) So assume that X_1 is $(\mathscr{Y}, \mathfrak{Z})$-good, A_1 is a $\mathfrak{Z}$-support of X_1, $A_1 \subseteq$

$\mathrm{Dom}(f), f \in \mathscr{F}$ and $X_2 = \{(G(g))(x) : x \in \mathrm{Dom}(g), f \restriction A \subseteq g\} \subseteq N$ and $(G(f))(x_1) = x_2$. First $x_1 \in X_1 \Rightarrow x_2 \in X_2$ by the definition of X_2. Second assume that $x_2 \in X_2$, so for some $y \in X_1$ and $g \in \mathscr{F}$ we have $f \restriction A_1 \subseteq g$ and $(G(g))(y) = x_2$. Let $B_2 \subseteq \mathrm{Dom}(g)$ be a $\mathfrak{Z}$-support of y. Let $B_1 \subseteq \mathrm{Dom}(f)$ be a $\mathfrak{Z}$-support of x_1; as $k \geq 3$ there is $f_1 \in \mathscr{F}$ such that $f \restriction (A_1 \cup B_1) \subseteq f_1$ and $g"(B_2) \subseteq \mathrm{Rang}(f_1)$ so $(G(f_1))(x_1) = x_2$. Now $(G(g^{-1} \circ f_1))(x_1)$ is well defined as $B_2 \subseteq \mathrm{Dom}(g^{-1} \circ f_1)$ hence is equal to $((G(g^{-1}) \circ (G(f_1)))(x_1) = y$ and $(g^{-1} \circ f_1) \restriction A_1 = \mathrm{id}_{A_1}$ hence $x_1 \in X_1 \equiv y \in X_1$ but $y \in X_1$ hence $x_1 \in X_1$ as required.

Clause (ii) So assume that $f \in \mathscr{F}, X$ is good and for $\ell = 1, 2$ the set $A_\ell \in I$ is a support of X and $A_\ell \subseteq \mathrm{Dom}(f)$. For $\ell = 1, 2$ let $X_\ell =: \{y \in N :$ for some $g \in \mathscr{F}$ and $y' \in X$ we have $f \restriction A_\ell \subseteq g$ and $(G(g))(y') = y\}$.
By the symmetry it is enough to show that $y \in X_1 \Rightarrow y \in X_2$. So assume $y \in X_1$ hence there are $g \in \mathscr{F}$ and $y' \in X$ such that $f \restriction A_1 \subseteq g$ and $(G(g))(y') = y$. As $y' \in X \subseteq N$, by Definition 2.6(1), clause (g) there is $B \in I$ which is a $\mathfrak{Z}$-support of y'. As $(G(g))(y') = y$ without loss of generality B is such that $B \subseteq \mathrm{Dom}(g)$ (see Definition 2.6(1), clause (f)). As $\mathscr{Y}$ is a k-system and $k \geq 3$ there is $f^* \in \mathscr{F}$ such that $f \restriction (A_1 \cup A_2) \subseteq f^*$ and $g"(B) \subseteq \mathrm{Rang}(f^*)$ hence for some $B^* \subseteq \mathrm{Dom}(f^*)$ we have $(f^*)"(B^*) = g"(B)$ so for some $y^* \in N$ we have $(G(f^*))(y^*) = y$. By clause (i) applied to A_2, f^*, y^*, y we have $y^* \in X \equiv y \in X_2$, so it is enough to prove that $y^* \in X$. Now easily $g^{-1} \circ f^* \in \mathscr{F}$, $B^* \subseteq \mathrm{Dom}(g^{-1} \circ f^*)$, $\mathrm{id}_{A_1} \subseteq g^{-1} \circ f^*$ and so $y^* \in \mathrm{Dom}(G(g^{-1} \circ f^*))$ and $(G(g^{-1} \circ f^*))(y^*) = y'$. Hence, as X is good (see Definition 2.13(1) clause (b)), we have $y^* \in X \equiv y' \in X$, but $y' \in X$ by its choice, so we are done.

Clause (iii) Easy, or see the proof of $\boxtimes(*)_1$ inside the proof of 2.16 below. $\square_{2.14}$

Now for the definition of successor for liftings of $\mathscr{Y}$, we naturally ask whether there is any.

2.15 Claim. *Assume $\mathscr{Y}$ is a k-system $k \geq 3$, Υ an inductive scheme and $\mathfrak{Z}$ is an $\mathbf{m}_1$-lifting of $\mathscr{Y}$ and $\mathbf{t} \in \mathbf{T}$.*
1) There is a $\bar{\varphi}^\Upsilon$-reasonable full $\mathbf{t}$-successor of $\mathfrak{Z}$ (and it is unique), similarly without $\mathbf{t}$.
2) $E_{\mathscr{Y},\mathfrak{Z}}$, defined in 2.13(4) is an equivalence relation on $\mathscr{P}_{Y,\mathfrak{Z}}$ (see Definition 2.13) and for every $f \in \mathscr{F}$, the function $G^+(f) : \mathscr{P}_{\mathscr{Y},\mathfrak{Z}} \to \mathscr{P}_{\mathscr{Y},\mathfrak{Z}}$ preverse the $E_{\mathscr{Y},\mathfrak{Z}}$-equivalence class.

Proof. 1) All is straight modulo part (2) (recalling 2.14).
2) Let $\mathfrak{Z}^*$ be the $\bar{\varphi}^\Upsilon$-reasonable full successor of $\mathfrak{Z}$ which exists by 2.14(2), and is a successor of $\mathfrak{Z}$ by 2.14, and is an $\mathbf{m}_1$-lifting by 2.16 below.

Why is $E_{\mathscr{Y},\mathfrak{Z}}$ an equivalence relation on $P_{\mathscr{Y},\mathfrak{Z}} = N^{\mathfrak{Z}^*} \backslash N^{\mathfrak{Z}}$? In short, by the properties of $\mathfrak{Z}$; in details:

$E_{\mathscr{Y},\mathfrak{Z}}$ is reflexive:
Let $X \in P_{\mathscr{Y},\mathfrak{Z}}$, so for some $A \in I$ we have $A\mathscr{R}^{\mathfrak{Z}^*}X$ (or equivalently $A\mathscr{R}X$) (see Definition 2.13(2)) and there is $f \in \mathscr{F}$ which is the identity on A, hence

(see Definition 2.13), $X \in \mathrm{Dom}(G^{\mathfrak{Z}^*}(f))$, and $(G^{\mathfrak{Z}^*}(f))(X) = X$ (as clause (f)) of Definition 2.6(1) holds as $\mathfrak{Z}^*$ is an $\mathbf{m}_1$-lifting of $\mathscr{Y}$.

$E_{\mathscr{Y},\mathfrak{Z}}$ is symmetric:
Use $G^{\mathfrak{Z}^*}(f^{-1}) = (G^{\mathfrak{Z}^*}(f))^{-1}$.

$E_{\mathscr{Y},\mathfrak{Z}}$ is transitive:
Use $G^{\mathfrak{Z}^*}(f_1 \circ f_2) \subseteq G^{\mathfrak{Z}^*}(f_1) \circ G^{\mathfrak{Z}^*}(f_2)$.
[This is similar to 2.4.]

By the definition of $E_{\mathscr{Y},\mathfrak{Z}}$ (see Definition 2.13(4)), clearly for $f \in \mathscr{F}$ the mapping $G^+(f) = G^{\mathfrak{Z}^*} \restriction \mathscr{P}_{\mathscr{Y},\mathfrak{Z}}$ preserves the $E_{\mathscr{Y},\mathfrak{Z}}$-equivalence class or use 2.16. $\square_{2.15}$

Clearly for "reasonable" cases, everything can be interpreted in $N^{\mathfrak{Z}^{\rm full}}_{\mathscr{Y},\mathfrak{t}}$, see later. We now prove that Definition 2.13(1)-(5) works as intended, i.e. any successor of $\mathfrak{Z}$ is an $\mathbf{m}_1$-lifting of $\mathscr{Y}$. In particular, we have to show that the functions defined are functions with the right domain and range and the E's are equivalence relations. This is included in the proof of 2.16.

2.16 Claim. *Assume $\mathscr{Y}$ is a k-system and $\mathfrak{Z}$ is an $\mathbf{m}_1$-lifting of $\mathscr{Y}$ (see Definition 2.1(2),Definition 2.6) and $k \geq 3$.*
Any successor $\mathfrak{Z}'$ of $\mathfrak{Z}$ is an $\mathbf{m}_1$-lifting of $\mathscr{Y}$.

Proof. We check the clauses in Definition 2.6. Let $G^+, G', R', N', \bar{P}'$ be as in Definition 2.13.

Clause (a): As N is transitive with M its set of urelements, and $X \in N' \backslash N \Rightarrow X \in \mathscr{P}_{\mathscr{Y},\mathfrak{Z}} \Rightarrow X \subseteq N$ also N' is transitive with M its set of urelements. Clearly N has the right vocabulary $\tau^+ = \tau_M \cup \{\in\}$ and $Q \in \tau_M \Rightarrow Q^{N'} = Q^M$. So N' is as required. Also $\bar{P}' = \langle P'_\ell : \ell < m_1 \rangle$, each P'_ℓ as required by $\mathbf{m}_1$.

Clause (b): By Definition 2.13(5)(c) we have that G' is $G^+ \restriction N'$ where the function G^+ is defined in part (3) of Definition 2.13 and $f \in \mathscr{F}$ implies $G^+(f)$ is a partial function with domain $\subseteq N \cup \mathscr{P}_{\mathscr{Y},\mathfrak{Z}}$ (see 2.14(1)). So G' is a function with domain $\mathscr{F}$ and $G'(f)$ by its definition is a partial function with domain $\subseteq N'$.

Clause (c):

Subclause (α): For $f \in \mathscr{F}$ we know that $G(f)$ is a function, $G(f) \restriction M = f$ (see Definition 2.6(1)), clause (c)(i)) and $G(f) = (G^+(f)) \restriction N$ (see Definition 2.13(3) particularly subclause (β), remembering that "X good $\Rightarrow X \notin N$" by Definition 2.13(1), clause (c)). As $M \subseteq N \subseteq N'$ and $G'(f) = G^+(f) \restriction N'$, together we get $f = (G^+(f)) \restriction M = (G'(f)) \restriction M$.

Subclause (β): Let $f \in \mathscr{F}$, $G(f) = f^1$, $G'(f) = f^2$ and let $x, y \in N'$ belongs to the domain of f' and we should prove

(α) $f^2(x) \in N'$

(β) $x \in N' \backslash N \Rightarrow f^2(x) \in N' \backslash N$

(γ) if $x \neq y$ are from N' then $f^2(x) \neq f^2(y)$

(δ) for every predicate $Q \in \tau_M, f^2$ preserve Q and $\neg Q$

(ε) $N' \models$ "$y \in x$" $\Leftrightarrow N' \models$ "$f^2(y) \in f^2(x)$"
(we shall do more toward proving clause (g) of Definition 2.6(1) below).

Note that for clause (α), as $f^2 \restriction N = G(f') \restriction N = G(f) = f^1$, it is enough to check it for $x \in N' \backslash N$, which is done in $\boxtimes, (*)_4 + (*)_1 + (*)_6$ below (as a good subset of N does not belong to N). Clause (β) also follows from $\boxtimes, (*)_4 + (*)_1 + (*)_6$ below. As for clause (γ), if $x, y \in N$ use $G(f') \restriction N = G(f)$; if $x \in N$ & $y \in N' \backslash N$ note that $G(f')(x) = (G(f))(x) \in N$ and $(G(f'))(y) \notin N$ by clause (β); similarly if $x \in N' \backslash N$ & $y \in N$; lastly if $x, y \in N' \backslash N$ we use clause (ε) proved below and N' being transitive (as $f'(x), f'(y)$ are subsets of N so $\notin M$). Now clause (δ) is easy as $G'(f) \restriction M = f$ and f being a partial automorphism of M being from $\mathscr{F}$.

Lastly, we consider clause (ε), so we let $x, y \in N'$. If $x \in N$, then $f^2(x)$ is necessarily in N too, but N is transitive, hence $N' \models$ "$y \in x$" $\Rightarrow y \in N$ and $N' \models$ "$z \in f^2(x)$" $\Rightarrow z \in N$, so as $f^2 \restriction N = f^1$ we are done. So we can assume $x \in N' \backslash N$, so x is a good subset of N, so for some $A_0 \in I, A_0 R^{3'} x$. We define

$\otimes_1$ $\quad z =: \{b \in N$:for some $g \in \mathscr{F}$ and $b' \in x$ we have
$f \restriction A_0 \subseteq g$ and $G(g)(b') = b\}$.

We need the following, and it suffices

$\boxtimes$ assume $x \in N' \backslash N$ and z is defined as in $\otimes_1$.
Then

$(*)_1$ z is a good subset of N with $A_1 =: f"(A_0)$ a support of z

$(*)_2$ $x = \{b' \in N$: for some $g \in \mathscr{F}$ and $b \in z$ we have
$f^{-1} \restriction (f"(A_0)) \subseteq g$ and $G(g)(b) = b'\}$

$(*)_3$ z does not belong to N

$(*)_4$ $z = f^2(x)$

$(*)_5$ if B is another $\mathfrak{Z}$-support of x, then $z' = z$ when $z' = \{b \in N$:
for some $g \in \mathscr{F}$ and $a \in x$ we have $f \restriction A \subseteq g$ and $G(g)(a) = b\}$.

$(*)_6$ $z \in N'$

Proof of $(*)_1$. We should check clauses (a),(b),(c) of Definition 2.13(1). Now clause (a) is trivial and clause (c) is dealt with in $(*)_3$ which we prove below (and we do not use it till then, so no vicious circle). So we concentrate on proving clause (b). So suppose:

(i) $a, b \in N$ and

(ii) $g_1 \in \mathscr{F}$ satisfies $A_1 \subseteq$ Dom(g_1) and $g_1 \restriction A_1$ is the identity and

(iii) $a \in$ Dom$[G(g_1)]$ and $b = G(g_1)(a)$.

Now we should prove that $a \in z \Leftrightarrow b \in z$. It is enough to prove $\Rightarrow$ as applying it to g_1^{-1} we get the other implication. As $b = G(g_1)(a)$ necessarily by clause (i) of Definition 2.6 for some $\mathfrak{Z}$-support B_1 of a we have $B_1 \subseteq \mathrm{Dom}(g_1)$.

Assume $a \in z$ then by the definition of z we can find $g \in \mathscr{F}$ and $a' \in X$ such that $f \restriction A_0 \subseteq g$ and $G(g)(a') = a$. By Definition 2.6(1), clause $(f)(\beta)$ there is $B_2 \in I$ such that B_2 is a $\mathfrak{Z}$-support of a' and $B_2 \subseteq \mathrm{Dom}(g)$. As $k \geq 3$ and as we can replace g by any g^* such that $g \restriction (A_0 \cup B_2) \subseteq g^* \in \mathscr{F}$,without loss of generality $B_1 \subseteq \mathrm{Rang}(g)$. So, possibly changing B_2 without loss of generality $B_1 = g"(B_2)$ (see clause (b) of Definition 2.6(1)).

Let $g' = g_1 \circ g$, so $A_0 \cup B_2 \subseteq \mathrm{Dom}(g'), g' \restriction A_0 = g \restriction A_0 = f \restriction A_0$.
[Why? As $g \restriction A_0 = f \restriction A_0, f"(A_0) = A_1$ and $g_1 \restriction A_1 = \mathrm{id}_{A_1}$; also $B_2 \subseteq \mathrm{Dom}(g)$ and $g"(B)$ is equal to B_1 which is $\subseteq \mathrm{Dom}(g_1)$]. Hence $a' \in \mathrm{Dom}(G(g'))$ and so $G(g')(a') = (G(g_1))\,(G(g)(a')) = (G(g_1))\,(a) = b$. (See Definition 2.6(1), clause (j).)

So g', a' witness $b \in z$; so $b \in z$ has been proved under the assumption $a \in z$. So by symmetry we have proved $a \in z \Leftrightarrow b \in z$.

Proof of $(*)_2$.

Call the set in the right side x'.

First assume that $a \in x$, so a has a $\mathfrak{Z}$-support B_1 hence for some $g_1 \in \mathscr{F}$ we have $A_0 \cup B_1 \subseteq \mathrm{Dom}(g_1)$ and $f \restriction A_0 \subseteq g_1$, hence $a \in \mathrm{Dom}(G(g_1))$ and let $b =: (G(g))(a)$, so $b \in z$ by the definition of z, also b has $\mathfrak{Z}$-support $B_2 =: g"(B_1)$. Let $g_2 = g_1^{-1}$ so $g_2 \in \mathscr{F}$ and $G(g_2) = G(g_1)^{-1}$ hence $(G(g_2))(b) = a$. Lastly, as $f \restriction A_0 \subseteq g_1$ clearly $f^{-1} \restriction (f"(A_0)) \subseteq g_2$. Together g_2, b witness that $a \in x'$. So we have proved $a \in x \Rightarrow a \in x'$.

Second, assume that $a \in x'$, so we have witnesses g, b for this, i.e. $g \in \mathscr{F}, b \in z, f^{-1} \restriction (f"(A_0)) \subseteq g$ and $(G(g))(b) = a$. So we can find $B_1 \subseteq \mathrm{Dom}(g)$ a $\mathfrak{Z}$-support of b, so $B_0 = g"(B_1)$ is a $\mathfrak{Z}$-support of a. As $b \in z$ there are witnesses for it, that is, there are $g_1 \in \mathscr{F}$ and $b' \in x$ such that $f \restriction A_0 \subseteq g_1$ and $(G(g_1))(b') = b$, hence $g_1^{-1} \in \mathscr{F}, G(g_1^{-1}) = (G(g_1))^{-1}$ so without loss of generality $B_1 \subseteq \mathrm{Rang}(g_1)$ and let $B_2 = (g_1^{-1})"(B_1)$, but B_1 is a $\mathfrak{Z}$-support of b hence B_2 is a $\mathfrak{Z}$-support of b'. Let $g' = g \circ g_1 \in \mathscr{F}$. Now $A_0 \subseteq \mathrm{Dom}(g_1)$ and $g"_1(A_0) = f"(A_0) \subseteq \mathrm{Dom}(g)$ hence $A_0 \subseteq \mathrm{Dom}(g')$, and as $g_1 \restriction A_0 = f \restriction A_0$, and $g \restriction f"(A_0) = f^{-1} \restriction f"(A_0)$ clearly $g' \restriction A_0 = \mathrm{id}_{A_0}$. Also $B_2 \subseteq \mathrm{Dom}(g_1)$, $B_1 = g"(B_2) \subseteq \mathrm{Dom}(g)$, hence $B_2 \subseteq \mathrm{Dom}(g')$ and $(G(g'))(b') = (G(g \circ g_1))(b') = G(g)((G(g_1))(b')) = (G(g))(b) = a$, but as $g' \restriction A_0 = \mathrm{id}_{A_0}$ and $(*)_1$ we have $b' \in x \Leftrightarrow (G(g'))(b') \in x$ which means $b' \in x \Leftrightarrow a \in x$. But we have chosen $b' \in x$ hence $a \in x$. So we have proved that $a \in x' \Rightarrow a \in x$. Thus finishing the proof of $(*)_2$.

Proof of $(*)_3$. If $z \in N$ there is $A^* \in I$ such that A^* is a $\mathfrak{Z}$-support of z.

Now there is $f_1 \in \mathscr{F}, f \restriction A_0 \subseteq f_1$ such that $A^* \subseteq \mathrm{Rang}(f_1)$. So $z_1 = G(f_1^{-1})(z)$ is well defined and by $(*)_2$ we can check that $\{b \in N : b \in z_1\} = x$; contradiction to "$x \notin N$".

Proof of $(*)_4$.
Should be clear.

Proof of $(*)_5$.
By 2.14(1).

Proof of $(*)_6$.
By clause (b) of 2.13(5).

We continue checking the clauses in Definition 2.6.

Clause (d):
Easy as for $f, g \in \mathscr{F}$ we have $f \subseteq g \Rightarrow G(f) \subseteq G(g)$.

Clause (i):
By the symmetry it is enough to show that $G'(f^{-1}) \subseteq G'(f)^{-1}$.
So let $(G'(f^{-1}))(x) = z$.

Now we know that both $G(f)$ and $G(f^{-1})$ maps N to N and $N'\backslash N$ to $N'\backslash N$ (that is when defined), so if $x \in N$ we have $z \in N$ and we use "$\mathfrak{Z}$ satisfies Definition 2.6 (1), clause (i)" to get $(G(f)^{-1})(z) = x$ as required. So assume $x \in N'\backslash N$ hence $z \in N'\backslash N$. By $\boxtimes(*)_4 + \otimes_1$ and $\boxtimes(*)_2$ we are done.

Clause (j):
Assume $x_0 \in \mathrm{Dom}(G'(f))$, hence x_0 has a $\mathfrak{Z}'$-support $A_0 \subseteq \mathrm{Dom}(f)$, so by the definition of $f_2 \circ f_1 = f$ we have $A_0 \subseteq \mathrm{Dom}(f_1)$ and $f"_1(A_0) \subseteq \mathrm{Dom}(f_2)$. So we have $x_0 \in \mathrm{Dom}(G'(f_1))$ and $x_1 =: (G'(f_1))(x_0)$ has $\mathfrak{Z}'$-support $A_1 =: f"_2(A_0)$. Similarly $x_2 =: (G'(f_2))(x_1)$ is well defined and has $\mathfrak{Z}'$-support $A_2 =: f"_1(A_1)$ which is $\subseteq \mathrm{Rang}(f_2 \circ f_1) = \mathrm{Rang}(f)$. Now we would like to show that $x_2 = (G'(f))(x_0)$; if $x_0 \in N$ this should be clear so assume that $x_0 \in N'\backslash N$ hence $x_1, x_2 \in N'\backslash N$. Let $x'_2 = (G'(f))(x_0)$, it is well defined as x_0 has $\mathfrak{Z}'$-support $A_0, A_0 \subseteq \mathrm{Dom}(f)$ and it suffices to prove that $x_2 = x'_2$. So let $y \in N$ and we shall prove that $(y \in x_2) \equiv (y \in x'_2)$.
Let B_2 be a $\mathfrak{Z}'$-support, equivalently $\mathfrak{Z}$-support of y and let $y_2 = y$. We can find $f'_2 \in \mathscr{F}$ such that $f'_2 \restriction A_1 = f_2 \restriction A_1$ and $B_2 \subseteq \mathrm{Rang}(f'_2)$ so as A_1 is a $\mathfrak{Z}'$-support of x_1, clearly $(G'(f'_2))(x_1) = (G'(f_2))(x_1) = x_2$. Let $B_1 = ((f'_2)^{-1})"(B_2)$. Also we can find $f'_1 \in \mathscr{F}$ such that $f'_1 \restriction A_0 = f_1 \restriction A_0$ and $B_1 \subseteq \mathrm{Rang}(f'_1)$ so as A_0 is a $\mathfrak{Z}'$-support of x_0 clearly $(G'(f'_2))(x_0) = x_1$ and let $B_0 = ((f'_1)^{-1})"(B_1)$. Let $y_1 =: (G'((f'_2)^{-1}))(y_2)$, so $y_1 \in N$ has $\mathfrak{Z}$-support B_1, and let $y_0 = (G'((f'_1)^{-1}))(y_1)$, so $y_0 \in N$ has $\mathfrak{Z}$-support B_0. As $G'(f'_2)$ maps x_1 to x_2 and y_1 to y_2 we have by clause $(c)(\beta)$ on $\mathfrak{Z}'$ which we have already proved that $(y_2 \in x_2) = (y_1 \in x_1)$. Similarly as $G'(f'_1)$ maps x_0 to x_1 and y_0 to y_1 we have by clause $(c)(\beta)$ that $(y_1 \in x_1) \equiv (y_0 \in x_0)$ so together $(y_2 \in x_2) \equiv (y_0 \in x_0)$. Now $f' = f'_2 \circ f'_1 \in \mathscr{F}$ and its domain include $A_0 \cup B_0$ and $G'(f')$ maps y_0 to y_2 (by clause (j) for $\mathfrak{Z}$!); also as x_0 is in its domain (as $A_0 \subseteq \mathrm{Dom}(f')$ is a $\mathfrak{Z}'$-support of x_0) and as $f \restriction A_0 = f' \restriction A_0$ we have $(G'(f'))(x_0) = (G'(f))(x_0)$ but the later is x'_2. So $(G'(f'))(x_0) = x'_2$, so as $y_0 \in \mathrm{Dom}(G'(f'))$ by clause (c) we have $y_0 \in x_0 \equiv (G'(f'))(y_0)) \in x'_2$ but $(G'(f'))(y_0) = y_2$ so $(y_0 \in x_0) \equiv (y_2 \in x'_2)$. As

earlier we have gotten $(y_2 \in x_2) \equiv (y_0 \in x_0)$ together $(y_2 \in x_2) \equiv (y_2 \in x'_2)$ but $y_2 = y$ so we are done.

Clause (e): See Definition of $R^{\mathfrak{Z}'}$ in Definition 2.13(5), clause (d).

Clause (f):

Subclause (α):
So assume $AR^{\mathfrak{Z}'}y, f \in \mathscr{F}$ and $A \subseteq \mathrm{Dom}(f)$. First, if $y \in N$ then we use $G'(f) \restriction N = G(f)$ and $\mathfrak{Z}$ satisfying Definition 2.6(1), clause $(f)(\alpha)$. Second, if $y \in N'\backslash N$ then $AR^{\mathfrak{Z}'}y$ means $A\mathscr{R}y$ and clearly y is a good subset of N and by the definition of $G'(f)(= G^+(f))$, necessarily $y \in \mathrm{Dom}(G'(f))$. If in addition $f \restriction A = \mathrm{id}_A$, we should prove that $(G'(f))(y) = y$. Now by $\boxtimes(*)_4$ apply to y, A instead x, A_0 we have

$$(G'(f))(y) = \{b \in N : \text{for some } g \in \mathscr{F} \text{ and } b' \in y \text{ we have } f \restriction A \subseteq g \text{ (i.e. } \mathrm{id}_A \subseteq g) \text{ and } G(g)(b') = b\}.$$

But as $A\mathscr{R}y$ we have:

$$\mathrm{id}_A \subseteq g \ \ \& \ \ b \in \mathrm{Dom}(G(g)) \Rightarrow [b \in y \equiv G(g)(b) \in y]$$

which means that $(G'(f))(y) = y$, as required.

Subclause (β)(of (f)):
So assume $f \in \mathscr{F}$ and $y \in \mathrm{Dom}(G'(f))$ (hence $y \in N'$). First, if $y \in N$, recall that $G'(f) \restriction N = G(f)$ and use $\mathfrak{Z}$ satisfying clause $(f)(\beta)$ of Definition 2.6(1) and

$\boxtimes_2 \ \ R' \restriction (I \times N) = R.$

Second, if $y \in N'\backslash N$ see the definition of $G'(f) = G^+(f)$ and R'.

Clause (g):
See the choice of $R', \mathscr{R}$.

Clause (h):
The new case is: assume $A \subseteq \mathrm{Dom}(f), A \in I, X \in \mathrm{Dom}(G'(f)), X \in N'\backslash N$. We have to show $AR^{\mathfrak{Z}'}X \Leftrightarrow f"(A)R^{\mathfrak{Z}'}(G'(f))(X)$; now by by clause (i) it is enough to prove the implication $\Leftarrow$.

Let $A^* =: f"(A)$ and $X^* =: (G'(f))(X)$, so we know that $A, A^* \in I$ and $X, X^* \in N'\backslash N$ and $A^*\mathscr{R}X^*$. We have to show that $A\mathscr{R}X$. If $\neg A\mathscr{R}X$, then we can find $g \in \mathscr{F}, g \restriction A = \mathrm{id}_A$, and $z_0 \in \mathrm{Dom}(G(g)), z_1 = G(g)(z_0)$, such that $z_0 \in X \equiv z_1 \notin X$. We can find $B_0 \in I$ such that $B_0 R z_0$ and $B_0 \subseteq \mathrm{Dom}(g)$ and let $B'_1 =: g"(B_0)$. We can find $f_1, f \restriction A \subseteq f_1, B_0 \cup B_1 \subseteq \mathrm{Dom}(f_1), f_1 \in \mathscr{F}$ and without loss of generality $f_1 = f$ and $\mathrm{Dom}(g) = A \cup B_0$. Let $g^* = f \circ g \circ f^{-1}, B^*_0 = f"(B_0)$ and $B^*_1 = f"(B_1)$. Clearly $B^*_0, B^*_1 \in I$ and $f^{-1}, g \circ f^{-1}, g^* = f \circ g \circ f^{-1} \in \mathscr{F}$. Also $B^*_0 \subseteq \mathrm{Dom}(f^{-1}), (f^{-1})"(B^*_0) = B_0 \subseteq \mathrm{Dom}(g), g"(B_0) = B_1$

and $f"(B_1) = B_1^*$ hence together $g^*(B_0^*) = B_1^*$. Let $z_0^* =: (G(f))(z_0), z_1^* = (G(f))(z_1)$, so $(G(f^{-1}))(z_0^*) = z_0, (G(g))(z_0) = z_1, (G(f))(z_1) = z_1^*$, and as B_0^* is $\mathfrak{Z}$-support of $z_0^*, B_0^* \subseteq \mathrm{Dom}(g^*)$ necessarily $(G(g^*))(z_0^*) = z_1^*$.

We can also show that $(z_0 \in X) \equiv (z_0^* \in X^*)$ and $(z_1 \in X) \equiv (z_1^* \in X^*)$ by clause $(c)(\beta)$ which we already proved so remembering $(z_0 \in X) \equiv (z_1 \notin X)$ we get a contradiction to "$A\mathscr{R}X^*$" which we have assumed.

Clause (k):
Trivial. $\square_{2.16}$

Well we have Υ-successors of candidates (in Definition 1.8, implicitly in Definition 1.1) and we have successors of $\mathbf{m}_1$-liftings $\mathfrak{Z}$ of $\mathscr{Y} = (M, I, \mathscr{F})$ where $\mathfrak{Z}$ has in it a candidate $(N^{\mathfrak{Z}}, \bar{P}^{\mathfrak{Z}})$.

Of course, we like to connect then, specifically show that true $(\Upsilon, \mathbf{t})$-successor of $\mathfrak{Z}$ exists. This is not always true, as Definition 1.8 can lead us to elements of $N' \backslash N$ with no support in I. In Definition 2.13 we restrict ourselves to elements with support in I, and we can change the definition in 1.1, 1.8 to have it, but it seems to me not so convincing for a logic. Rather we show that the dichotomy assumptions (as in 2.1(3), 2.3) help.

When we use $\mathscr{L}^* = \mathscr{L}_{\mathrm{f.o.}}$, then dealing with Υ-successor is easier, we have to look less carefully at cardinalities, still we need a dichotomy property (see Definition 2.6) in order to get a $\mathfrak{Z}$-support to every member.

2.17 Claim. *Assume that:*

- (a) $\mathscr{Y} = (M, I, \mathscr{F})$ *is a* k*-system,* $k \geq 3$
- (b) Υ *is an inductive scheme for* $\mathscr{L}_{\mathrm{f.o.}}(\tau_M^+)$
- (c) *for every* $\ell < m_0^\Upsilon$ *any subformula of* ψ_ℓ^Υ *has* $\leq k$ *free variables; also for every* $\ell < m_1^\Upsilon$ *any subformula of* φ_ℓ^Υ *has* $\leq k$ *free variables*
- (d) $\mathbf{t} \in \mathbf{T}$
- (e) $\mathscr{Y}$ *is a* $\mathbf{t}$*-dichotomical* k*-system*
- (f) *for every* $\ell < m_0^\Upsilon$ *the formula* ψ_ℓ^Υ *has* $\leq k/2$ *free variables.*

Then

- (α) *if* $\mathfrak{Z}$ *is an* $\mathbf{m}_1^\Upsilon$*-lifting of* Y *then* $\mathfrak{Z}$ *has a true* $(\Upsilon, \mathbf{t})$*-successor (see Definition 2.13(9))*
- (β) *for every* $t \leq \infty$ *there is an* $\mathbf{m}_1^\Upsilon$*-lifting* $\mathfrak{Z}^t$ *of* Y *such that recalling Definition 1.1(3A), we have* $(N^{\mathfrak{Z}^t}, \bar{P}^{\mathfrak{Z}^t}) = (N_t[M, \Upsilon, \mathbf{t}], \bar{P}_t[M, \Upsilon, \mathbf{t}])$
- (γ) *if* $t \leq t_\iota[M, \Upsilon, \mathbf{t}]$ *and* $\iota \in \{1, \ldots, 7\}$ *or* $t \leq t_\iota[M, \Upsilon, \mathbf{t}]$ & $\iota = 11, \ldots, 17$ *then there is an* $\mathbf{m}_1^\Upsilon$*-lifting* $\mathfrak{Z}^{\mathbf{t}}$ *of* $\mathscr{Y}$ *such that* $(N^{\mathfrak{Z}^t}, \bar{P}^{\mathfrak{Z}^t})$ *is equal to*

$$(N_t[M, \Upsilon], \bar{P}_t[M, \Upsilon]) \quad \text{or} \quad (N_t[M, \Upsilon, \mathbf{t}], \bar{P}_t[M, \Upsilon, \mathbf{t}]),$$

respectively; similalry for $\iota = 21, \ldots, 27$.

Proof. Clearly clause (β) follows from clause (α), just prove by induction on t. Also clause (γ) follows from (β) and clause (α), so we deal with clause (α). Let $\mathfrak{Z} = (N, \bar{P}, G, R)$.

The main point is to prove

$\boxtimes$ assume $\ell < m_0^{\Upsilon}$ and for $\bar{a} \in {}^{\ell g(\bar{y})}N$ and $X = X_{\bar{a}} = \{b \in N : (N, \bar{P}) = \psi_\ell[b, \bar{a}]\}$. Then $A \mathscr{R} X$ for some $A \in I$ or $\{\{b \in N : (N, \bar{c}) \models \psi_\ell[b, \bar{a}']\} : \bar{a}' \in {}^{\ell g(\bar{y})}N\}| > \mathbf{t}(\|M\|)$.

Let $m = \ell g(\bar{y})$, so $2(m+1) \leq k$ and $\bar{a} = \langle a_\ell : \ell < m \rangle$, let $A_\ell \in I$ be a $\mathfrak{Z}$-support of a_ℓ and let $\bar{b}_\ell$ be a list of A_ℓ without repetitions, and define a function h^* with domain $[m]$ with $h^*(\ell) = \bar{b}_\ell$. We define a 2-place relation E on $h^*/E^0_{I,m}$, see Definition 2.1(3)(γ):

$\oplus_1$ if for $j = 1, 2, f_j \in \mathscr{F}$ and $\bigcup\limits_{\ell \in [m]} \mathrm{Rang}(h^*(\ell)) \subseteq \mathrm{Dom}(f_j)$ and $h_j = f_j * h^*$ then $h_1 E h_2 \Leftrightarrow X_{G(f_1)(\bar{a})} = X_{G(f_2)(\bar{a})}$ (where $G(f_j)(\langle a_\ell : \ell < m \rangle) = \langle G(f_j)(a_\ell) : \ell < m \rangle$.

Now

$\oplus_2$ E belongs to $\mathscr{E}_{I,m}(\emptyset)$, see Def 2.1(3)$(\delta)$.

Why? by traslating this means that:

$(*)$ if $\bar{a}^1, \bar{a}^2, \bar{a}^3, \bar{a}^4 \in \{(G(f))(\bar{a}) : \bigcup\limits_{\ell < m} A_\ell \subseteq \mathrm{Dom}(f)$ and $f \in \mathscr{F}\}$, and $(G(f))(\bar{a}^1{}^\frown \bar{a}^2) = \bar{a}^3{}^\frown \bar{a}^4$, then $N \models \psi(\bar{a}^1, \bar{a}^2) \equiv \psi(\bar{a}^3, \bar{a}^4)$

where $\psi(\bar{y}_1, \bar{y}^2) =: (\forall x)[\varphi(x, \bar{y}_1) \equiv \varphi(x, \bar{y}_2)]$.

Now every subformula of $\psi(\bar{y}_1, \bar{y}_2)$ has at most $2m + 1 < k$ free variables or is a subformula of $\varphi(x, \bar{y})$ hence has $\leq k$ free variables, hence by 2.9 we have $N \models \psi(\bar{a}^1, \bar{a}^2)$ iff $N \models \psi(\bar{a}^3, \bar{a}^4)$, so we are done showing $\oplus_2$.

Now $m \leq k/2$ and we are assuming that $\mathscr{Y}$ is a $\mathbf{t}$-dichotomical k-system, so $(\beta)_1$ or $(\beta)_2$ of Definition 2.1(4) holds. Now $(\beta)_2$ gives the desirable first possible conclusion in $\boxtimes$ and $(\beta)_1$ gives that $|\{b : N \models \psi_\ell(b, \bar{a}') : \bar{a}' \in {}^m N\}| > \mathbf{t}(\|M\|)$, hence second possible conclusion in $\boxtimes$. $\square_{2.17}$

2.18 Claim. *1) Assume that*

- (a) *$\mathscr{Y}$ is a counting k-system with $k \geq 3$*
- (b) *Υ is an inductive scheme, in $\mathscr{L}_{\mathrm{f.o.}}$ or in $\mathscr{L}_{\mathrm{f.o.\ +\ na}}$ or $\mathscr{L}_{\mathrm{card}}$ or $\mathscr{L}_{\mathrm{card},\mathbf{T}}$*
- (c) *for $\ell < m_0^{\Upsilon}$, every subformula of $\psi_\ell(y, \bar{x})$ has at most $k-1$ free variables and for $\ell < m_1^{\Upsilon}$ every subformula of φ_ℓ has at most $k-1$ free variables*
- (d) *$\mathbf{t} \in \mathbf{T}$*
- (e) *$\mathscr{Y}$ is a $\mathbf{t}$-dichotomical k-system*

(f) *for* $\ell < m_0^{\Upsilon}$ *the formula* ψ_ℓ *has* $\le k/2$ *free variables.*
Then
- (α) *if* $\mathfrak{Z}$ *is an* $\mathbf{m}_1^{\Upsilon}$*-lifting, then* $\mathfrak{Z}$ *has a true* $(\mathbf{t}, \Upsilon)$*-successor*
- (β) *for every* t *there is an* $\mathbf{m}_1^{\Upsilon}$*-lifting* $\mathfrak{Z}^t$ *such that* $(N^{\mathfrak{Z}^t}, \bar{P}^{\mathfrak{Z}^t}) = (N_t[M, \Upsilon, \mathbf{t}], \bar{P}_t[M, \Upsilon, \mathbf{t}])$.

Proof. The proof is as in 2.17 , but we know by 2.10 that the partial automorphism $G(f)$ preserves also $\psi_1(y, \bar{x})$ and even $(\exists^k y)\psi_1(y, \bar{x})$ when every subformula of ψ_1 has $< k$ free variables; (note that only now having $2m+1 < k$ rather than $2m+1 \le k$ seem helpful or repeat the proof of 2.10 as we can use the $< k$ just for subformulas of φ). $\square_{2.18}$

2.19 Remark. Why do we still need in 2.18 the "$\mathbf{t}$-dichotomical"? Just to guarantee that the true $(\mathbf{t}, \Upsilon)$-successor is included in the full one.

2.20 Claim. *Assume*

- (a) $\mathscr{Y}$ *is a* k*-system with* $k \ge 3$
- (b) Υ *is an inductive scheme in* $\mathscr{L}_{\text{f.o.}}$
- (c) *for* $\ell < m_0^{\Upsilon}$*, every subformula of* $\psi_\ell(y, \bar{x})$ *has at most* $k-1$ *free variables and for* $\ell < m_1^{\Upsilon}$ *every subformula of* φ_ℓ *has at most* $k-1$ *free variables*
- (d) $\mathbf{t} \in \mathbf{T}$
- (e) $\mathscr{Y}$ *is medium* $\mathbf{t}$*-dichotomical*
- (f) *for* $\ell < m_0^{\Upsilon}$ *the formula* ψ_ℓ *has* $\le k/2$ *free variables.*

Then: the conclusion of 2.18 holds, so if $t \le t_\iota(M, \Upsilon, \mathbf{t})$*, then for some* $\mathbf{m}_1^{\Upsilon}$*-lifting* $\mathfrak{Z}^t$ *we have* $(N^{\mathfrak{Z}^t}, \bar{P}^{\mathfrak{Z}^t}) = (N_t[M, \Upsilon, \mathbf{t}], \bar{P}_t[M, \Upsilon, \mathbf{t}])$.

Proof. Like the proof of 2.17, 2.18.

2.21 Definition. 1) We say that $\mathscr{H}$ is a witness to the k-equivalence of $\mathscr{Y}_1$ and $\mathscr{Y}_2$ if

- (a) for $\ell = 1, 2$ we have $\mathscr{Y}_\ell = (M_\ell, I_\ell, \mathscr{F}_\ell)$ is a k-system
- (b) $\mathscr{H}$ is a family of partial isomorphisms from M_1 into M_2
- (c) for every $g \in \mathscr{H}$, we have $\mathrm{Dom}(g) \in I_1$, $\mathrm{Rang}(g) \in I_2$
- (d) if $g \in \mathscr{H}$ and $f_1 \in \mathscr{F}_1$ then $g \circ f_1 \in \mathscr{H}$
- (e) if $g \in \mathscr{H}$ and $f_2 \in \mathscr{F}_2$ then $f_2 \circ g \in \mathscr{H}$
- (f) if $g \in \mathscr{H}$ and $A \in I_1[k-1]$ and $B \in I_1$, then for some $g_1 \in \mathscr{H}$ we have $g \restriction A \subseteq g_1$ and $B \subseteq \mathrm{Dom}(g_1)$
- (g) if $g \in \mathscr{H}$ and $A \in I_2[k-1]$ and $B \in I_2$, then for some $g_1 \in \mathscr{H}$ we have $g^{-1} \restriction A \subseteq g_1^{-1}$ and $B \subseteq \mathrm{Rang}(g_1)$.

2) We say that $\mathscr{H}$ is a witness to the dichotomical (k,r)-equivalence of $(\mathscr{Y}_1,\mathbf{t}_1)$ and $(\mathscr{Y}_2,\mathbf{t}_2)$ if

- (i) $\mathscr{Y}_\ell$ is a $(\mathbf{t}_\ell,k,r)$-dichotomical k-system for $\ell=1,2$
- (ii) $\mathscr{H}$ is a witness to the k-equivalence of $\mathscr{Y}_1$ and $\mathscr{Y}_2$
- (iii) each $g\in\mathscr{H}$ preserved the possibility chosen in the definition of $(\mathbf{t},k,r)$-dichotomical.

If we omit r, we mean $s=[k/2]$.
3) We say that $\mathscr{H}$ is a witness to the counting k-equivalence of $\mathscr{Y}_1$ and $\mathscr{Y}_2$ if

- (i) $\mathscr{Y}_\ell$ is a counting $(\mathbf{t},k,r)$-system for $\ell=1,2$
- (ii) $\mathscr{H}$ is a witness to the k-equivalence of $\mathscr{Y}_1$ and $\mathscr{Y}_2$
- (iii) each $g\in\mathscr{H}$ preserve the cardinalities involved in the definition of "counting $(\mathbf{t},k,r)$-system".

4) Similarly with "medium dichotomy".

2.22 Main Conclusion: Assume

- (a) $\mathscr{Y}_\ell=(M_\ell,I_\ell,\mathscr{F}_\ell)$ is a $\mathbf{t}_\ell$-dichotomical k-system, and $\tau(M_\ell)=\tau$ for $\ell=1,2$
- (b) $\mathscr{H}$ is a witness to the k-equivalence of $\mathscr{Y}_1$ and $\mathscr{Y}_2$
- (c) $\chi\in\mathscr{L}_{\rm f.o.}(\tau^+)$, i.e. a first order sentence in the vocabulary $\tau^+=\tau\cup\{\in\}$,
- (d) Υ is an inductive scheme for $\mathscr{L}_{\rm f.o.}$
- (e) every subformula of χ and of ψ_ℓ^Υ and of φ_ℓ^Υ has at most $<k$ free variables
- (f) $\mathbf{t}_1,\mathbf{t}_2\in\mathbf{T}$
- (g) every formula ψ_ℓ^Υ has $\le k/2$ free variables and $k\ge 3$ of course.

Then

- (α) let[4] $\iota\in\{2,3,4,5\}$; the truth value of $\theta_{\Upsilon,\chi,\mathbf{t}_1}$ in M_1 and $\theta_{\Upsilon,\chi,\mathbf{t}_2}$ in M_2 under $\models_\iota$ are equal except possibly when: for some $\ell\in\{1,2\}$ we have the truth value of $\theta_{\Upsilon,\chi,\mathbf{t}_\ell}$ in M_ℓ is undefined whereas that of $\theta_{\Upsilon,\chi,\mathbf{t}_{3-\ell}}$ in $M_{3-\ell}$ is well defined and $t_\iota[M_\ell,\Upsilon,\mathbf{t}_\ell]<t_\iota[M_{3-\ell},\Upsilon,\mathbf{t}_{3-\ell}]$.
- (β) For any t, if $N^\ell=N_t[M_\ell,\Upsilon,\mathbf{t}_\ell]$ is well defined for $\ell=1,2$, then for every sentence $\theta\in\mathscr{L}_{\rm f.o.}(\tau^+)$ such that every subformula has at most k free variables, we have $N^1\models\theta\Leftrightarrow N^2\models\theta$
- (γ) if $\iota\in\{2,3,4,5\}$ and $\theta^\ell=\theta_{\Upsilon,\chi,\mathbf{t}^\ell}\in\mathscr{L}_\iota^{\mathbf{T}}(\mathscr{L}_{\rm f.o.}(\tau))$ is ι-good for $\ell=1,2$: then $M_1\models_\iota$ "θ^1" iff $M_2\models_\iota$ "θ^2"
- (δ) for any t, if $N^\ell=N_t[M_\ell,\Upsilon,\mathbf{t}_\ell]$ is well defined for $\ell=1,2$ and $\theta=\theta(x_1,\dots,x_n)\in\mathscr{L}_{\rm f.o.}(\tau^+)$ is a formula such that every subformula has at most k free variables then:
  - $\oplus$ let $a_1,\dots,a_m\in N^1$ and $g\in\mathscr{H}$, then $N^1\models_\iota\theta[a_1,\dots,a_m]$ iff $N^2\models_\iota\theta[(G(f))(a_1),\dots,(G(f))(a_m)]$.

[4] Here and below, for $\iota\in\{6,7\}$ the conclusion is similar but expressed more cumbersomely

Proof. First we can prove clause (δ) by induction on the quantifier depth of θ, as in 2.9.

Second, note that clause (α) follows from clause (β).

Third, note that clause (β) follows from clause (δ)
and the definition of satisfaction 1.1.

Lastly, concerning clause (γ) follows the definition of $\mathscr{L}$-good (and clause (β)). $\square_{2.22}$

2.23 Conclusion. Assume

(a) $\mathscr{Y}_\ell = (M_\ell, I_\ell, \mathscr{F}_\ell)$ is a counting k-system, $\tau(M_\ell) = \tau$ for $\ell = 1, 2$

(b) $\mathscr{H}$ is a witness for the counting k-equivalence of $\mathscr{Y}_1$ and $\mathscr{Y}_2$

(c) $\chi \in L_{\mathrm{card},\mathbf{T}}$

(d) Υ is an inductive scheme for $\mathscr{L}^* = \mathscr{L}_{\mathrm{card},T}(\tau^+)$ or $\mathscr{L}_{\mathrm{card}}(\tau^+)$

(e) every subformula of χ and of ψ_ℓ^Υ (for $\ell < m_0^\Upsilon$) and of φ_ℓ^Υ (for $\ell < m_1^\Upsilon$) has at most $k-1$ free variables

(f) $\mathbf{t} \in \mathbf{T}$

(g) if $\ell < m_0^\Upsilon$ then ψ_ℓ^Υ has $\leq k/2$ free variables and $k \geq 3$ of course.

Then

(α) if $\theta = \theta_{\Upsilon,\chi,\mathbf{t}}$ and $\iota \in \{2, 3, 4, 5, 6, 7, 11, 22\}$ then $M_1 \models_\iota \theta \Leftrightarrow M_2 \models_\iota \theta$ and $M_1 \models_\iota \neg\theta \Leftrightarrow M_2 \models_\iota \neg\theta$

(β) For any t if $N^\ell = N_t[M_\ell, \Upsilon, \mathbf{t}]$ is well defined for $\ell = 1, 2$, then for every sentence $\theta \in \mathscr{L}_{\mathrm{f.o.}}(\tau^+)$ such that every subformula has at most $k-1$ free variables, we have $N^1 \models_\iota \theta \Leftrightarrow N^2 \models_\iota \theta$

(γ) for any t, if $N^\ell = N_t[M_\ell, \Upsilon, \mathbf{t}]$ are well defined (for $\ell = 1, 2$), and $\theta = \theta(x_1, \ldots, x_n) \in \mathscr{L}_{\mathrm{f.o.}}(\tau^+)$ is a formula such that every subformula has at most $k-1$ free variables we have:

$\oplus$ if $a_1, \ldots, a_m \in N^1$ and $g \in \mathscr{H}$, each $(G(f))(a_\ell)$ is well defined then $N^1 \models_\iota \theta[a_1, \ldots, a_m]$ iff $N^2 \models_\iota \theta[(G(f))(a_1), \ldots, (G(f))(a_m)]$.

Proof. Straight.

Now

2.24 Conclusion. Assume

(a) $\mathscr{Y}_\ell = (M_\ell, I_\ell, \mathscr{F}_\ell)$ is a medium $\mathbf{t}$-dichotomical k-system, $\tau(M_\ell) = \tau$ for $\ell = 1, 2$

(b) $\mathscr{H}$ is a witness for the medium $\mathbf{t}$-dichotomical k-equivalence of $\mathscr{Y}_1$ and $\mathscr{Y}_2$

(c) $\chi \in \mathscr{L}_{\mathrm{f.o.}}(\tau^+)$

(d) Υ is an inductive scheme for $\mathscr{L}^* = \mathscr{L}_{\mathrm{f.o.}}(\tau^+)$

(e) every subformula of χ and of ψ^{Υ}_{ℓ} (for $\ell < m^{\Upsilon}_0$) and of $\varphi^{\Upsilon}_{\ell}$ (for $\ell < m^{\Upsilon}_1$) has at most $k-1$ free variables

(f) $\mathbf{t}_\ell \in \mathbf{T}$

(g) if $\ell < m^{\Upsilon}_0$ then ψ^{Υ}_{ℓ} has $\leq k/2$ free variables and $k \geq 3$ of course.

Then

(α) if $\theta = \theta_{\Upsilon,\chi,\mathbf{t}}$ and $\iota \in \{1,\ldots,5,6,7,11,22\}$ then $M_1 \models_\iota \theta \Rightarrow M_2 \models_\iota \theta$ and $M_1 \models_\iota \neg\theta \Leftrightarrow M_2 \models_\iota \neg\theta$

(β) For any t if $N^\ell = N_t[M_\ell, \Upsilon, \mathbf{t}_\ell]$ is well defined for $\ell = 1,2$, then for every sentence $\theta \in \mathscr{L}_{\mathrm{f.o.}}(\tau^+)$ such that every subformula has at most $(k-1)$-free variables, we have $N^1 \models \theta \Leftrightarrow N^2 \models \theta$.

Proof. Straightforward.

2.25 Discussion We consider now some variants.
1) We have to consider the stopping times. If $\mathscr{L}^* = \mathscr{L}_{\mathrm{car},\mathbf{T}}$ or $\mathscr{L}_{\mathrm{card},\mathbf{T}}$ this is natural, (and they are stronger logics than the earlier variants). If we still would like to analyze in particular for the others, we should be careful how much information can be gotten by the time.
2) We can modify Υ such that in N_{t+1} we can reconstruct the sequence $\langle (N_\ell, \bar{P}_\ell) : \ell \leq s\rangle$ (see §4).
3) We can change our presentation: first proving the equivalences for $N^{\mathfrak{Z}[\mathscr{Y}_\ell,\mathbf{t}_\ell]}$ for $\ell = 1,2$, (see Definition 5.5) and then proving that $(N_t[M^{Y_\ell}, \Upsilon, \mathbf{t}], \bar{P}_t[M, \Upsilon, \mathbf{t}])$ is interpretable in $N^{\mathfrak{Z}^{\mathrm{full}}_{\mathscr{Y}_\ell,\mathbf{t}}}$ uniformly.

§3 The Canonical Example

We apply §2 to the canonical example: random enough graph.

3.1 Definition. Let τ be a fixed (finite) vocabulary consisting of predicates only. We say M is a $(\mathbf{s},k)$-random τ-model if every quantifier free 1-type over $A \subseteq M, |A| < k$ (not explicitly inconsistent) is realized in M by at least $\mathbf{s}(\|M\|)$ elements. If $\mathbf{s}$ is constantly ∞ we may write k-random.

Remark. We can restrict the set of allowable quantifier free types if it is nice enough e.g. R two-place symmetric irreflexive. More generally see e.g. [BlSh 528].

3.2 Definition. $\mathbf{T}_{\mathrm{pol}}$ is $\mathbf{T}_{\mathbb{Q}}$, where for a set $Q \subseteq \mathbb{R}$ containing an unbounded set of reals > 0 let $\mathbf{T}_Q$ be $\{f_q : q \in Q, q > 0\}$ where $f_q : \omega \to \omega$ is $f_q(n) = n^q$, or more exactly, $\lceil n^q \rceil$ the least integer $\geq n^q$.

3.3 Claim. *Assume*

(a) q^*, k *are integers* > 1 *and* $k^* = q^* k$

(b) $s \leq k, s > 0$ *integer*

(c) M_ℓ *is* $(\mathbf{s}_\ell, 3k^*)$*-random* τ*-model for* $\ell = 1, 2$

(d) $\mathbf{t}_\ell(\|M_\ell\|) < (\mathbf{s}_\ell(\|M_\ell\|))^{q^*+1}/(q^*+1)!$ *and* $\mathbf{s}_\ell(\|M_\ell\|) > q^*$

(e) Υ *is an inductive scheme for* $\mathscr{L}_{\rm f.o.}(\tau_{[2]})$, χ *a sentence in* $\mathscr{L}_{\rm f.o.}(\tau_{[2]})$ *and each subformula of any formula among* $\psi^\Upsilon_\ell\ (\ell < m^\Upsilon_0), \varphi^\Upsilon_m (m < m^\Upsilon_1)$ *and* χ *has at most* s *free variables*

Then

$\oplus$ *if* $\iota \in \{2, 3, 4, 5, 6\}$ *and* $\theta_{\Upsilon,\chi,\mathbf{t}_\ell}$ *is* ι*-good (at least for* M_ℓ *that is, the truth values below are defined) for* $\ell = 1, 2$, *then* $M_1 \models_\iota \theta_{\Upsilon,\chi,\mathbf{t}_1} \Leftrightarrow M_2 \models_\iota \theta_{\Upsilon,\chi,\mathbf{t}_2}$.

3.4 Remark. 1) Compare clause (β) with 2.21(2).
2) Why in 3.3 do we use clause (d)? As there we use $N_t[M_\ell, \Upsilon, \mathbf{t}_\ell]$ so for some t we may add the sets in $\mathscr{P}_t[M_1, \Upsilon, \mathbf{t}_1]$ to $N_t[M_1, \Upsilon, \mathbf{t}_1]$ (in defining $N_{t+1}[M_1, \Upsilon, \mathbf{t}_1]$ but do not add $\mathscr{P}_\ell[M_2, \Upsilon, \mathbf{t}_2]$ to $N_t[M_2, \Upsilon, \mathbf{t}_2]$ in defining $N_{t+1}[M_2, \Upsilon, \mathbf{t}_2]$.
3) We concentrate on ι-good sentences (or local versions) in order to have neat results. Otherwise we have really to be more careful, e.g. about the cardinalities of the $N_t[M, \Upsilon, \mathbf{t}]$'s. This is very reasonable for counting logic.
4) We have ignored in this claim, and others in this section, the cases $10 < \iota$. We can deal with them, if we note the following required changes. We have to note that the function $\mathbf{t}$ is split to two functions; one, $\mathbf{t}^{sp}$, for telling us how to increase N_t to N_{t+1}, that is which additional families of subsets of N_t are allowed to be added, and for this function the parallel of clause (d) should be demanded. Secondly we have to consider what families are added in each stage (so for the counting and the medium analog our situation may be better).

Proof. We let $I_\ell = \{A \subseteq M_\ell : |A| \leq q^*\}$ and

$$\mathscr{F}_\ell = \{f : f \text{ is a partial automorphism of } M_\ell \text{ and } \mathrm{Dom}(f) \text{ has } \leq q^* k \text{ elements}\}$$

$(*)_1$ $\mathscr{Y}_\ell = (M_\ell, I_\ell, \mathscr{F}_\ell)$ is a k-system
[why? the least obvious clause in Definition 2.1(1) is clause (D) which holds by Definition 3.1 above.]

$(*)_2$ $\mathscr{Y}_\ell = (M_\ell, I_\ell, \mathscr{F}_\ell)$ is $(\mathbf{t}_\ell, s)$-dichotomical, (see Def 2.1(4)).

Why? The proof of $(*)_2$ takes most of the proof. Let $m \in \mathbb{N}$ be $1 \leq m \leq s$ and let E be an equivalence relation from $\mathscr{E}^0_{I,m}(\emptyset)$ so it is an equivalence relation on $h^*/E^0_{\mathscr{Y}_\ell,m}$ where $h^* : [m] \to \mathrm{Seq}_{I_\ell}, E$ satisfying $(*)$ of clause (δ) of Definition 2.1(3). For $h \in h^*/E^0_{\mathscr{Y}_\ell,m}$ let $\bar{b}_h$ be the concatenation of $h(1), h(2), \ldots, h(m)$.

Without loss of generality h^* is one-to-one and even $\bar{b}_{h^*}$ is with no repetitions. Let $t^* = \ell g(\bar{b}_{h^*})$ so $t^* \le q^* k \le k^*$. By clause (c) and the definition of $\mathscr{F}$ there is a quantifier free formula $\varphi(\bar{x}, \bar{y}) \in \mathscr{L}(\tau)$ with $\ell g(\bar{x}) = \ell g(\bar{y}) = t^*$ such that $h_1 E h_2$ iff $M_\ell \models \varphi[\bar{b}_{h_1}, \bar{b}_{h_2}]$.

Clearly without loss of generality $\varphi(\bar{x}, \bar{y})$ tells us that the quantifier free type of $\bar{x}$ and of $\bar{y}$ (same as that of $\bar{b}_{h^*} = h^*(1)^\frown h^*(2)^\frown \ldots {}^\frown h^*(m)$), call it $p(\bar{x})$ and let $p[M] = \{\bar{a} \in {}^{t^*}M : \bar{a} \text{ realizes } p(\bar{x})\}$, so we can look at E restricted to this set. Can there be $\bar{a}_0, \bar{a}_1 \in {}^{t^*}(M_\ell)$ realizing the same quantifier free type $p(\bar{x})$ (over the empty set) which are not E-equivalent? If not, then $|p[M_\ell]/E| = 1$ and we are done so assume there are. We can find $\bar{a}_2 \in {}^{t^*}(M_\ell)$ realizing the same quantifier free type $p(\bar{x})$ and disjoint to $\bar{a}_0{}^\frown \bar{a}_1$ (use "M_ℓ is $(\mathbf{s}_\ell, 3k^*)$-random"), so $\bar{a}_2, \bar{a}_j$ are not E-equivalent for some $j \in \{0,1\}$; so without loss of generality $\bar{a}_0, \bar{a}_1$ are disjoint. Now we ask "are there disjoint $\bar{b}_0, \bar{b}_1 \in {}^{t^*}(M_\ell)$ realizing $p(\bar{x})$ which are E-equivalent"? If yes, we easily get a contradiction to "E an equivalence relation" (by finding $\bar{b}'$, a sequence from M_ℓ realizing $p(\bar{x})$ such that both $\bar{b}'{}^\frown \bar{a}_0, \bar{b}'{}^\frown \bar{a}_1$ realize the same quantifier free type as $\bar{b}_0{}^\frown \bar{b}_1$; contradiction). So: no disjoint $\bar{b}_0, \bar{b}_1 \in p[M_\ell]$ are E-equivalent. Next we claim that

$\bigotimes$ for some $u \subseteq [0, t^*)$ and subgroup $\mathbf{g}$ of the group of permutations of u, moreover of $\mathbf{g}^* = \mathbf{g}^*_u = \{\sigma \in \ \mathrm{Per}(u)$: if $\bar{a} \in {}^{t^*}(M_\ell)$ realizes $p(\bar{x})$ then $\langle a_{\sigma(i)} : i \in u\rangle$ realizes the same quantifier type as $\bar{a} \restriction u\}$, we have:
$E \restriction \{\bar{a} \in {}^{t^*}(M_\ell) : \bar{a} \text{ realizes } p(\bar{x})\} = E_{\mathbf{g}} \restriction \{\bar{a} \in {}^{t^*}(M_\ell) : \bar{a} \text{ realizes } p(\bar{x})\}$ where for any subgroup $\mathbf{g}'$ of $\mathbf{g}^*, E_{\mathbf{g}'} = E^{t^*, M_\ell}_{\mathbf{g}'}$ is defined by: for $\bar{a}, \bar{b} \in {}^{t^*}(M_\ell), \bar{a} E_{\mathbf{g}'} \bar{b} \Leftrightarrow (\exists \sigma \in \mathbf{g})(\langle a_{\sigma(t)} : t \in u\rangle = \langle b_r : t \in u\rangle)$ (this is an equivalence relation).

[Why? It is enough to show: assume $\bar{a}, \bar{b}, \bar{c} \in {}^{t^*}(M_\ell)$ realize $p(\bar{x})$ and $\bar{a}, \bar{b}$ are E-equivalent, $r^* < t^*$ and $a_{r^*} \notin \{b_t : t < t^*\}$ then $\bar{c}/E$ does not depend on c_{r^*} (i.e. if $\bar{c}' \in {}^{t^*}(M_\ell)$ realizes $p(\bar{x})$ and $r' < t^*$ & $r' \ne r^* \Rightarrow c_{r'} = c'_{r'}$ then $\bar{c} E \bar{c}'$.)
Toward this end, let σ be the partial function from $[0, t^*)$ to $[0, t^*)$ such that $\sigma(r_1) = r_2 \Leftrightarrow a_{r_1} = b_{r_2}$. Clearly σ is one to one and $r^* \notin \mathrm{Dom}(\sigma)$.
We choose by induction on $j^* \le t!$ a sequence $\bar{a}^j \in p[M_\ell]$ such that $\bar{a}^0 = \bar{a}, \bar{a}^1 = \bar{b}$, the quantifier free type of $\bar{a}^j{}^\frown \bar{a}^{j+1}$ in M_ℓ is the same as the quantifier free type of $\bar{a}^0{}^\frown \bar{a}^1 = \bar{a}{}^\frown \bar{b}$ in M_ℓ and $(\forall r)[a^{j+1}_r \notin \{a^j_r : r < t^*\} \Rightarrow a^j_r \notin \{a_r : r < t^*\}]$. Clearly $j < t^*! \Rightarrow \bar{a}^j E \bar{a}^{j+1}$, hence $j \le t^*! \Rightarrow \bar{a} E \bar{a}^j$. Let σ^j be the partial function from $[0, t^*)$ to $[0, t^*)$ defined by $\sigma^j(r_1) = r_2 \Leftrightarrow a_{r_1} = a^j_{r_2}$. Clearly $\sigma^0 = \mathrm{id}_{[0,t^*)}$ and $\sigma^{j+1} = \sigma \circ \sigma^j$. Clearly $\sigma^{t^*!}$ is the identity function on some subset of $[0, t^*)$ and $a^{t^*!}_{r_2} = a^0_{r_1} (= a_{r_1}) \Leftrightarrow r_1 = r_2$ & $\sigma^{t^*!}(r_1) = r_1$. Now given $\bar{c}'$ as above we can find $\bar{b} \in p[M_1]$ such that $\bar{c}{}^\frown \bar{b}$ and $\bar{c}'{}^\frown \bar{b}$ realizes the same quantifier free type as $\bar{a}^0{}^\frown \bar{a}^{t!}$, hence $\bar{c} E \bar{b}$ & $\bar{c}' E \bar{b}$ hence $\bar{c} E \bar{c}'$. Easily we are done proving

$\boxtimes$ $p[M_\ell]/E$ has cardinality $\ge (\mathbf{s}_\ell(\|M_\ell\|)^{|u|} \cdot (|\mathbf{g}^*|/|\mathbf{g}|)$.

[Why? Clear by $\boxtimes$ and Definition 3.1.]
This number is $\ge (\mathbf{s}_\ell(\|M_\ell\|)^{|u|}/|u|!$. Hence if $|u| > q^*$ as $\mathbf{s}_\ell(\|M\|) > k^* = q^* k \ge |u| > q^*$ (see assumption (d)) the number is $\ge (\mathbf{s}_\ell(\|M_\ell\|))^{q^*+1}/(q^*+1)!$ which

by assumption (d) is $> \mathbf{t}_\ell(\|M_\ell\|)$ and we get one of the allowable answers in Definition 2.1(4). So we can assume that $|u| \leq q^*$ and this gives the second possibility so we have finished proving $(*)_2$.

Let

$$\mathscr{H} = \big\{ f : f \text{ is a partial embedding of } M_1 \text{ into } M_2 \text{ with } \mathrm{Dom}(f) \text{ having } \leq q^* k \text{ members} \big\}.$$

$(*)_3$ $\mathscr{H}$ is a (k,s)-witness to the equivalence of $(\mathscr{Y}_1, \mathbf{t}_1)$ and $(\mathscr{Y}_2, \mathbf{t}_2)$ [why? straight.]

So we can apply 2.22 and get the desired result. $\square_{3.3}$

Lastly, we can conclude the answer for the question in [BGSh 533]:

3.5 Theorem. *1) Assume $\iota \in \{2,3,4,5\}, \tau = \{R\}, R$ binary symmetric irreflexive, $\mathbf{p} \in (0,1)$ and $\mathbf{T}$ are given and each $\mathbf{t} \in \mathbf{T}$ is bounded by a polynomial. The logic $\mathscr{L}^{\mathbf{T}}_\iota(\mathscr{L}_{\mathrm{f.o.}})(\tau)$ satisfies the undecided 0-1 law for finite random enough model, that is graph with a fix probability $\mathbf{p} \in (0,1)$ which means; if $\theta_1 = \theta_{\Upsilon,\chi,\mathbf{t}} \in \mathscr{L}^{\mathbf{T}}_\iota(\mathscr{L}_{\mathrm{f.o.}})(\tau)$ and $\theta_0 = \theta_{\Upsilon,\neg\chi,\mathbf{t}}$ then $\langle Min\{Prob(\mathscr{M}_n \models \theta_0), Prob(\mathscr{M}_n \models \theta_1)\} : n < \omega\rangle$ converge to zero[5], where $\mathscr{M}_n$ is $G(n,\mathbf{p})$, the random graph on n with edge probability $\mathbf{p}$.*
2) Moreover also the undecided$^+$ $0-1$-law hold; which means:
if $\theta_1 = \theta_{\Upsilon,\chi,\mathbf{t}} \in \mathscr{L}^{\mathbf{T}}_\iota(\mathscr{L}_{\mathrm{f.o.}})(\tau)$ and $\theta_0 = \theta_{\Upsilon,\neg\chi,\mathbf{t}}$ then for some $\ell \in \{0,1\}$ the sequence $\langle \{Prob(\mathscr{M}_n \models \theta_\ell) : n < \omega\}$ converges to zero,
3) Similarly for any fixed (finite) vocabulary τ consisting of predicates only $\bar{\mathbf{p}} = \langle \mathbf{p}_R : R \in \tau\rangle, \mathbf{p}_R \in (0,1)_{\mathbb{R}}$.

Proof. 1) Let $\theta_0, \theta_1, \Upsilon, \chi, \mathbf{t}$ be as above and $\varepsilon > 0$. Let s be large enough such that assumption (e) of 3.3 holds, choose $k = 3s$ so assumption (b) there holds. We choose $\mathbf{s}_\ell$ as $\mathbf{s}(n) = (n-k) \times (\mathrm{Min}\{\mathbf{p}^k/2, (1-\mathbf{p})^k/2\})$ and let q^* be integer > 0 such that for n large enough $\mathbf{s}(n)^{q^*+1} \geq \mathbf{t}(n)(q^*+1)!$ and let $k^* = q^* k$.
Let n be large enough and $\mathscr{M}^1_n, \mathscr{M}^2_n$ be random enough (for $G(n,\mathbf{p})$). We would like to apply Claim 3.3 with $M_1 = \mathscr{M}^1_n, M_2 = \mathscr{M}^2_n$ and Υ, χ as in the definition of θ_0, θ_1 and $\mathbf{t}_1 = \mathbf{t}_2 = \mathbf{t}$ and $\mathbf{s}_1 = \mathbf{s}_2$ large enough. This is straight, noting that the case of the truth value of θ_1 in $\mathscr{M}_n$ is undefined, i.e. we run out of resources, just help us.
2) Similarly, this time for n is large enough, $n_1 \geq n, n_2 \geq n$ and $\mathscr{M}^1_{n_1}, \mathscr{M}^2_{n_2}$ are random enough (for $G(n_1,\mathbf{p}), G(n_2,\mathbf{p})$ respectively).
3) Similarly $\square_{3.5}$

3.6 Discussion: 1) It is reasonable to consider the undecided law if we know that the $(N_t[M_\ell, \Upsilon, \mathbf{t}_\ell], \bar{P}_t[M_\ell, \Upsilon, \mathbf{t}_\ell])$ for $\ell = 1,2$ are quite equivalent for every

[5] We do not ask that for some $\ell < 2$ the probability for the satisfaction of θ_ℓ converges to 1, as the decision when to stop may be complicated. If we e.g. use an inside "clock" to tell us when to stop, this disappears

t, when $\|M_1\| = \|M_2\|$, but we do not have information otherwise.
2) We might prefer to have the usual zero-one law. There are some avenues to get (at some price), see also 3.7,3.8 below; we may consider all sentences, $\iota \in \{2, \ldots, 5\}$ and the usual 0-1 law.

We have to try to use $\mathbf{t}$ which tries to diagonalize the right sets. That is, using 3.3 for $\mathbf{t}_1 = \mathbf{t}_2 = \mathbf{t}$, we can get strong enough equivalence of $N^+_t[M_1, \Upsilon], N^+_t[M_2, \Upsilon]$, which is fine if $\|M_1\| = \|M_2\| < \|M_2\|$, so it is enough if $N^+_{t_1}[M_\ell, \Upsilon], N^+_{t_2}[M_\ell, \Upsilon]$ with $t_1 = t_\iota[M_1, \Upsilon, \mathbf{t}_1], t_2 = t_\iota[M_2, \Upsilon, t_1]$ and choose $\mathbf{t}$ such that they are quite equivalent. As in $N^+_t[M_\ell, \Upsilon]$ we can define $\mathbb{N} \restriction \{0, \ldots, t-1\}$, this requires $\mathbf{t}(\|M_\ell\|)$ to be quite large compare to $\|M_\ell\|$. So we can get our desired 0-1 laws and all possible ι's, but for a logic remote from our intention.

On the other hand, we may restrict our family of sentences (here)

3.7 Theorem. *If in Theorem 3.5 we restrict ourselves to the good sentences, i.e. the logic is $\mathscr{L}^{\mathbf{T}}_\iota(\mathscr{L}_{\text{f.o.}})^{\text{good}}$ and $\iota \in \{2, 3, 4, 5\}$, then the usual 0-1 law holds.*

Proof. Similar. $\square_{3.7}$

3.8 Theorem. *1) Assume $\iota \in \{2, 3, 4, 5\}, \tau = \{R\}, R$ binary symmetric irreflexive predicate, $\mathbf{p} \in (0, 1)_{\mathbb{R}}$ and $\mathbf{T}$ are given and for each $\mathbf{t} \in \mathbf{T}$ for some integer r and $\varepsilon \in (0, 1/2)_{\mathbb{R}}$ we have $0 = lim\langle \mathbf{t}(n)/n^{r+1-\varepsilon} : n \in \mathbb{N}\rangle$ and $\infty = lim\langle \mathbf{t}(n)/n^{r+\varepsilon} : n \in \mathbb{N}\rangle$. Then the logic $\mathscr{L}^{\mathbf{T}}_\iota(\mathscr{L}_{\text{f.o.}})(\tau)$ satisfies the results in 3.3 - 3.7.*
2) Similarly for any fixed (finite vocabulary τ consisting of predicates only, $\bar{p} = \langle \mathbf{p}_R : R \in \tau\rangle, \mathbf{p}_R \in (0, 1)_{\mathbb{R}}$.

Proof. 1) Suppose that $M^* \models \theta_{\Upsilon,\chi,\mathbf{t}}$ and this because in stage t^* the run stop, i.e., in a good way; and assume further that M is random enough graph (for our given Υ and χ). We can find E_ℓ for $\ell < \ell^*$ such that E_ℓ is a quantifier free formula with $2m_\ell$ variable defining an equivalence relation on $p_\ell(M)$ for every random enough graph M, $p_\ell(\bar{x})$ a complete quantifier free type with m_ℓ variables said to be pairwise distinct. We can find non negative integers r^t_ℓ for $\ell < \ell^*, t \le t^*$ such that: if M is random enough graph and $N_t = N_t[M, \Upsilon]$ then $\|N_t\| = \Sigma_{\ell<\ell^*} r^t_\ell |({}^{m_\ell}M)/E_\ell|$. Now the expected value of $|({}^{m_\ell}M)/E_\ell|$ is of the form $\mathbf{p}' \times \text{binomial}(n, m_\ell)$ for some constant $\mathbf{p}'$. The distribution is similar enough to normal (see [Sh 550]) to ensure that the run on $\mathscr{M}_n$ will not stop for $t \le t^*$ for over using resources.
2) Similarly $\square_{3.8}$

What we have done for random graphs we can do to unary predicate. The point is to replace claim 3.3 by a parallel one (the rest will follow).

3.9 Claim. *1) Assume that the vocabulary τ is $\{P\}$. P a unary predicate and*

(a) *q^+, q^-, k are integers > 1 and h is a decreasing (not necessarily strictly) function from $\{0, 1, \ldots, q^+\}$ to $\{0, 1, \ldots, q^-\}$,*

(b) *s is an integer $\leq k$ but > 0*

(c) *$M_\ell = (|M_\ell|, P^{M_\ell})$ for $\ell = 1, 2$*

(d) *if $q \leq q^+$ then*

(i) $\mathbf{t}_\ell(\|M_\ell\|)$ *is at least* $|P^{M_\ell}|^q \times |(M_\ell \setminus P^{M_\ell})|^{h(q)}$

(ii) $\mathbf{t}_\ell(\|M_\ell\|)$ *is strictly smaller than*

$$\text{binomial}(|P^{M_\ell}|, q+1) \times \text{binomial}(|M_\ell \setminus P^{M_\ell}|, h(q)),$$

(iii) $\mathbf{t}_\ell(\|M_\ell\|)$ *is strictly smaller than*

$$\text{binomial}(|P^{M_\ell}|, q) \times \text{binomial}(|M_\ell \setminus P^{M_\ell}|, h(q)+1).$$

2) The parallel of 3.5- 3.8 holds

Proof. Similar but letting $I_\ell = \{A \subseteq M_\ell$: for some $q \leq q^+$ we have $|A \cap P^{M_\ell}| \leq q$ and $|A \setminus P^{M_\ell}| \leq h(q)\}$. $\square_{3.9}$

3.10 Definition. 1) We say M is a τ-model with k-elimination of quantifiers if for every subsets A_0, A_1 of M, $|A_0| = |A_1| < k$ and an isomorphism f from $M \restriction A_0$ onto $M \restriction A_1$ and $a_0 \in M$ there is $a_1 \in M$ such that $f = f \cup \{\langle a_0, a_1\rangle\}$ is an isomorphism from $M \restriction (A_0 \cup \{a_0\})$ onto $M \restriction (A_1 \cup \{a_1\})$.
2) We replace "quantifiers" by "quantifier and counting" if we add: and the two sets $\{a'_0 \in M : a'_0, a_0$ realize the same quantifier free type over $A_0\}$ and $\{a'_1 \in M : a'_1, a_1$ realize the same quantifier free type over $A_1\}$ has the same number of elements (we can then get it to equivalence relations on m-tuples).

3.11 Claim. *1) Assume (a), (b), (e) in 3.3 replacing first order by counting logic and*

(c)$^-$ (α) *M_ℓ are τ-models which has k-elimination of quantifier for $\ell = 1, 2$*

(β) *if $\varphi(\bar{x}, \bar{y})$ is a quantifier free formula defining an equivalence relation and $\ell g(\bar{x}) = \ell g(\bar{y}) \leq k^*$ then the number of classes is $> \mathbf{t}_\ell(M_\ell)$ for $\ell = 1, 2$ or for some $u \subseteq [0, \ell g(\bar{x}))$ with $\leq q^*$ elements, some $\varphi'(\bar{x} \restriction u, \bar{y} \restriction u)$ defines the equivalence relation in M_1 and in M_2*

(γ) *if $2r + s \leq k$ and for $\ell = 1, 2$, $\bar{a}_\ell \in {}^s(M_\ell), \bar{x}^j = \langle x^j_i : i < s\rangle$, for $j = 1, 2$,*
$\varphi_\ell = \varphi_\ell(\bar{x}^1, \bar{x}^2, , \bar{a}_\ell)$ is first order and defines in M_0 an equivalence relation E_ℓ and $\varphi_1 = \varphi_2$ and the quantifier free types of $\bar{a}_1$ in M_1 and $\bar{a}_2$ in M_2 are equal, then $|^r(M_1)/E_\ell|, |^r(M_2)/E_2|$ are equal or $|^r(M_1)/E_1| > \mathbf{t}_1(\|M_1\|)$ & $|^r(M_2)/E_2| > \mathbf{t}_2(\|M_2\|)$.

Then

$\oplus$ *if* $\iota \in \{1-7, 11-17, 21-27\}$ *then* $M_1 \models_\iota \theta_{\Upsilon,\chi,\mathbf{t}_1} \Leftrightarrow M_2 \models_\iota \theta_{\Upsilon,\chi,\mathbf{t}_2}$.

2) We have a theorem like 3.3, 3.8 (for ι *as above) using 3.11(1) instead of 3.3.*
3) We can in parts (1), (2) and in 3.3, 3.5, 3.7, 3.8 replace $\mathscr{L}_{\text{f.o.}}$ *by* $\mathscr{L}_{\text{f.o. + na}}$.

Proof. Straight.

3.12 Claim. *1) Choiceless polynomial time does not capture counting logic.*
2) Similarly for the pair $(\mathscr{L}^{\mathbf{T}}_\iota(\mathscr{L}_{\text{f.o.}}), \mathscr{L}^{\mathbf{T}}_\iota(\mathscr{L}_{\text{f.o.+na}}))$*, the pair* $(\mathscr{L}^{\mathbf{T}}_\iota(\mathscr{L}_{\text{f.o.+na}}),$ $\mathscr{L}^{\mathbf{T}}_\iota(\mathscr{L}_{\text{card}}))$ *and the pair* $(\mathscr{L}^{\mathbf{T}}_\iota(\mathscr{L}_{\text{f.o.+na}}), \mathscr{L}^{\mathbf{T}}_\iota(\mathscr{L}_{\text{card},\mathbf{T}}))$.
3) We can apply 3.11 to show that the pairs (M^1_n, M^2_n) *of models from [GuSh 526] are not distinguished by our logics (for a sentence* θ *for* n *large enough.*

Proof. 1) Use 2.22, 3.9 on the question: $|P^M| \geq \|M\|/2$ with $\tau = \{P\}$.
2),3) Also easy $\square_{3.12}$

3.13 Remark. : Y. Gurevich asks for 0-1 laws, as in 3.3 - 3.8, for the general framework of §2. The answer is quite straight by 3.14, 3.15, when we use constant I.

3.14 Definition. Let τ be a fixed vocabulary consisting of predicates only. We say M is $(\mathbf{s}, k)$-random model if: every quantifier free 1-type over $A \subseteq M, |A| < k$ (not explicitly inconsistent) is realized in M by at least $\mathbf{s}(\|M\|)$ elements.

We are, of course, using

3.15 Claim. *Let* $k, s > 0$ *be integers, let* $\tau = \{R\}, R$ *symmetric irreflexive and* $\mathbf{p} \in (0, \frac{1}{2}]_{\mathbb{R}}$. *The probability that: for* $\mathscr{M}_n$ *a* $G(n; \mathbf{p})$ *random graph (so set of vertices in* $[n]$*),* $(\mathscr{M}_n, I)$ *is not* (s, k)*-random is* $\leq \Sigma_{\ell<k}\text{binomial}(n, \ell)\times$ Prob*(flipping* $n - \ell$ *times a coin with probability* $\mathbf{p}/2^\ell$ *for a head we get* $< s$ *heads).*

§4 Relating the Definitions in [BGSh 533] to the One Here

4.1 Discussion If we just like to replace the creation of $N_t[M, \Upsilon, \mathbf{t}]$ by ASM, we can note that we can straightforwardly code the actions of the ASM by a monotone Υ; the waste is small except that we are not allowed to omit old elements so for fine measurements this make a difference. But we can just replace $N_t[M, \Upsilon, \mathbf{t}]$ by a situation of the ASM with no lose and no real difference in the proofs. Still, the reader may instead of just accepting or understanding this observation choose to read the formal translation below. Though this seem trivial, writing the details of a translation is tedious.

4.2 Discussion: How do we relate between the definitions above and [BGSh 533]?

(*i*) an infinite structure I there corresponds to a τ-model here

(*ii*) a state A there corresponds to a model of the form $N = N_t[M, \Upsilon]$ in 1.3 and $(N, \bar{P})$ in 1.8

(*iii*) dynamic function there corresponds to $P_{t,\ell}$ here

(*iv*) an object x is active at A in 5.1 there, corresponds to $x \in N$

(*v*) a program 5.7 there corresponds to an Υ in 1.1(2) here (mainly the first order formulas used);

(*vi*) the counting function in 5.5 there corresponds to the cardinality quantifier (1.1(6)) here

(*vii*) the polynomial functions p, q in 5.1 there corresponds to $\mathbf{t}^{\rm sp}, \mathbf{t}^{\rm tm} \in \mathbf{T}$ here

(*viii*) the logic there corresponds to $\mathscr{L}^{\mathbf{T}}_{\iota}(\mathscr{L}_*), \mathscr{L}_* \in \{\mathscr{L}_{\rm f.o.}, \mathscr{L}_{\rm r.o.+\ na}, \mathscr{L}_{\rm card}\}$ here.

If we insist on P_ℓ being individual constants this still can be done with a price. The $P_{t+1,\ell}$ can in the usual set theory manner be actually 7-place function from N_t to N_t or 7-place relation on N_t, or be the universe of N_t. Understanding this to interpret the successor step there to here we need that all parts of the program are expressible in $\mathscr{L}_{\rm f.o.}$ (or $\mathscr{L}_{\rm card}$). For the other direction we need to show f.o. operations can be expressed by the programs of ASM there (see 6.1 there), no problem (and not needed to show our results solved problems there).

4.3 Lemma. *1) Let π be a program concerned with τ-models (in [BGSh 533, 4.7]'s sense). We can find a natural number $r^* \geq 1$ and Υ such that:*

$\boxtimes_1(a)$ *Υ is an inductive scheme in $\mathscr{L}_{\rm f.o.}$ with $\tau^\Upsilon = \tau^\pi$*

(*b*) *for every integer polynomials $p(n), q(n)$ and τ-model M such that $p(n) \geq 2, q(n) > n + 2$ we have $M \models \theta_{\Upsilon, p^*(n), q^*(n)}$ (in the sense of Definition 1.3) iff $M \models \bar{\pi}$ (in the sense of [BGSh 533]) when*

(*) $p^*(n) = 3 + r^* p(n), q^*(n) = 2 + 2q(n)$.

2) If $\bar{\pi}$ is as in [BGSh 533, §11], that is with being able to compute the cardinality of M (= set of atoms) then a similar result holds but Υ is in $\mathscr{L}_{\rm f.o.\ +\ na}$.
3) If $\bar{\pi}$ is as in [BGSh 533] for cardinality logic (see §4 there), then a similar result holds but Υ is in $\mathscr{L}_{\rm card}$ and spaces do not use $\theta_{\Upsilon, 3+n+r^*(p), 2+n+2q(n)}$.
4) We can replace p, q by arbitrary function from $\mathscr{T}$ and get similar results.

Before proving 4.3:

4.4 Observation. 1) If we identify truth, false with the sets $\emptyset, |M|$ and $\mathbf{m}_1$ is as in 1.1, then the state (for π, i.e. Υ there, etc.) of [BGSh 533] are the same $(M, \mathbf{m}_1)^+$-candidate here when we identify a state there with its set of active elements (O.K., as they carry the same information). Sometimes we use any transition set $\subseteq V_\infty(M)$ containing the active member.

The main point is

4.5 Claim. *Let π be a program concerned with τ-models and the states correspond to $(M, \mathbf{m}_1)$-candidates.*
1) For every term σ for some natural number $r(\sigma)$ (actually at most its depth) and pure inductive scheme Υ which is monotonic and $\psi_0 = [y = x_0]$ we have:

$\boxtimes_1$ *if $(N_i, \bar{P}_i)$ is a $(M, \mathbf{m}_1)$-candidate for $i = 0, \ldots, r(\sigma)$ and $(N_{i+1}, \bar{P}_{i+1})$ is the (M, Υ)-successor of $(N_i, \bar{P}_i)$ and $\bar{\zeta}$ is a variable assignment of $\bar{x}$, a sequence listing the free variables of σ into N_σ, then the interpretation from σ under $\bar{\zeta}$ in N_i satisfies $d =: val_{N_i,\zeta}(\sigma) \in N_{r(\sigma)}$.*

2) Moreover in (1) we can find also formulas $\langle \psi_{\ell,r}(x, \bar{y}_\ell, \bar{z}) : \ell < m_1^\Upsilon, r < r(\sigma)\rangle$ such that

$\boxtimes_2$ *in $\boxtimes_1$ above we can add:*
if $r \le r(\sigma)$ and let $N'_r = N_0 \cup (TC(d) \cap N_r)$ where TC is transitive closure, then $N'_{r+1} = N_r \cup \{a :$ for some $\bar{b} \in {}^{\ell g(\bar{y})}N_r, (N'_r, P) \models \psi_{\ell,2}(a, \bar{b}, \bar{\zeta})\}$ (identifying $\bar{\zeta}$ with a sequence of members of N_0).

3) For every rule $\mathbf{R}$ (see [BGSh 533, 4.5]) there are $r(\mathbf{R}) \in \mathbb{N}$ and an inductive scheme Υ in $\mathscr{L}_{\text{f.o.}}$ such that (P_0 is zero place relation):

$\boxtimes_3$ *if $(N_i, \bar{P}_i)$ is a (M, Υ)-candidate for $i \le r(\mathbf{R})$ and $(N_{i+1}, \bar{P}_{i+1})$ is the Υ-successor of $(N_i, \bar{P}_i)$ for $i < r(\mathbf{R})$ and $P_{i,0} =$ truth, then $i < r(\mathbf{R}) \Rightarrow N_i \subseteq N_{i+1}$, the stationary $(N_{i(\mathbf{R})}, \bar{P}^-_{i(\mathbf{R})})$ is the $\mathbf{R}$-udpate of $(N_0, \bar{P}^-_0)$, $P_{i,r(\mathbf{R})} =$ truth where $\bar{P}^- = \bar{P} \restriction [1, m_1^\Upsilon)$.*

4) In (3) we can even have

$\boxtimes_4$ $N_r = N_0 \cup \{x : x$ *active in* $N_{r^*}\}$.

Proof. 1) By induction on the term.
2) As N_{r+2} can be (uniformly) interpreted in N_r.
3) By induction on the rule $\mathbf{R}$. $\square_{4.5}$

Proof of 4.3.

We describe what is an Υ-successor rather than let $r^* = r(\mathbf{R}^\pi) + 1, \mathbf{R}^\pi$ is the rule which π is. Then say formally what is Υ.

Now the predicates (and function symbols) $\{P_k^\Upsilon : k < m_0^\Upsilon\}$ serve some purposes:

kind 1: The dynamic predicate and function symbols of π, say P_k for $k \in w(1)$, say $P_{k(1,0)}$ will denote $\emptyset$, $P_{k(1,a)}$ will denote the set of atoms.

For notation simplicity

kind 2: $P_{k(2,0)}$ unary predicate will serve to denote the set of active elements; and

kind 3: $P_{k(3,1)}, \ldots, P_{k(3,r^*)}$ will be zero place relations, they will denote the time modulo r^*, say for $t = 0$ they are all false; for $t = 1$ we get true, true false ..., for $t = 2$ we get true, true, true, false... (without loss of generality $r^* \geq 3$ for $t = 3 + r^*s + r, r < r^*$ we have $P_{k(3,r')} \equiv r' = r$). The reason is that in our translation one step for $\bar{\pi}$ will be translated to r^* steps in the construction of the N_t's and the translation begins only with $t = 1$. Now we can describe almost a translation.

Now Υ is such that:

(a) $N_1[M, \Upsilon]$ is $M \cup \{M, \emptyset\}$,
$N_2[M, \Upsilon]$ is $M_1[M, \Upsilon] \cup \{1\}$, recall $1 = \{\emptyset\}$
$N_3[N, \Upsilon]$ is $N_1[M, \Upsilon] \cup \{1, 2\}$,

(b) if $t^* = 3 + r^*s$, then $P_{t^*,k(3,0)} =$ true, $P_{t^*,k(3,1)}, \ldots, c_{t^*,k(3,r^*)}$ are false and we take care that $P_{t^*+r,k(3,r')} = r' = 0 \bmod r^*$ for $r' \leq r^*$ and $\langle (N_{t^*+2}, \bar{P}_{t^*+r}) : i \leq r(\mathbf{R}^*) \rangle$ is as in 4.4(3)+(4). Moreover $\bar{P}_{t^*+r^*} = \bar{P}_{t^*+r(\mathbf{R}^*)}$ and $N_{t^*+r^*}$ is the set of active members of $(N_{t^*+r(\mathbf{R}^*)}, \bar{P}_{t^*+r(\mathbf{R}^*)})$.

Well, as in $\theta = \theta_{\Upsilon,1+r^*p,q}$ and $\iota = 1$, the stopping decision for time will be the same, but we still have to deal with the space (up to now using $r + r^*p$ would be O.K.). However, between t^* and $t^* + r^*$ we have to preserve N_{t^*} till creating $N_{t^*+r(\mathbf{R}^*)}$ and only then can we omit the elements of N_{t^*} no longer necessary.

So we will have

kind 4: individual constants $P_{k(4,1)}, P_{k(4,2)}$

(c) if $t^* = 3 + r^*s$ then: $P_{t,k(4,0)} < P_{t,k(4,1)} < P_{t,k(4.2)}$ are the three last ordinals, $P_{t,k(4,0)}$ is an active ordinal but not $P_{t,k(4,1)}, P_{t,k(4,2)}$ and $x \in N_{t^*}$ is active iff $P_{t,k(4,2)} \cup x \in N_{t^*}$.

So $\|N_{t^*}\| = 2 + 2\|\{x \in N_{t^*} : x \text{ active}\}\|$.

Now starting with N_{t^*} in deciding N_{t^*} we omit and non-active elements except the two last ordinals and then do as earlier by 4.4(3)+(4) for $r = 1, \ldots, r(\mathbf{R}^\pi)$ taking care to have the right natural numbers.

So defining $N_{i^*+2^*}$, we take care of the "doubling".

2), 3), 4) Similarly. $\square_{4.3}$

Less critical is the inverse relation

4.6 Lemma. *1) Let Υ be an inductive scheme for $\mathscr{L}_{\text{f.o.}}, \iota = 7, \tau = \tau^\Upsilon, \chi \in \mathscr{L}_{\text{f.o.}}(\tau)$.*
Then we can find $r^* \geq 1$ and a program π for the same vocabulary as in [BGSh 533, §4] such that for every integer polynomials $p(n), q(n)$ and τ-model M such that:

$M \models_\iota \theta_{\Upsilon,\chi,p,q}$ *(see Definition 1.3) iff* $M \models \bar{\pi}$ *where* $\bar{\pi} = (\pi, p^*, q^*)$ *and* $\models$ *is as in [BGSh 533, §4] where:* $p^*(n) = r^*, p^*(n) + r^*, q^*(n) = q(n) + 2$

4.7 Lemma. *1) If $\bar{\pi}$ is as in [BGSh 533, §11], that is with being able to compute the cardinality of M (= set of atoms) then a similar result holds but Υ is in $\mathscr{L}_{\text{f.o. + na}}$.*
2) If $\bar{\pi}$ is as in [BGSh 533] for cardinality logic (see §4 there), then a similar result holds but Υ is in $\mathscr{L}_{\text{card}}$ and spaces do not use $\theta_{\Upsilon,3+n+r^*(p),2+n+2q(n)}$.
3) We can replace p, q by arbitrary function from T and get similar results.

Proof. 1) We ignore too short runs for simplicity. The "$q(n) = q(n) + 2$" comes from [BGSh 533] starting with two extra elements so during the computation we preserve having two entry elements (except when we notice we are to stop-see below).

Now every step of the computation for $\bar{\pi}$ is translated to r^* step during the computation of the $N_t[M, \Upsilon]$'s.
What do we do in those r^* steps? First we compute the relations on N_t definable first order subformulas of the ψ_ℓ. We also translate χ to be equivalent to what should be in the next state and then add the new elements (so $N_{t+1} \models \chi$ was computed in N_t, as in 4.4).
2), 3), 4) Similar to part (1) + 4.3.

[How do we "compute" the first order formulas? Where $P_{\varphi(\bar{x})}$ code $\varphi(\bar{x})$, we of course represent all subformulas and do it inductively.]

Atomic are given

negation is by "if $P_{\varphi(\bar{x})}(\bar{a})$ = truth then $P_{\varphi(\bar{x})}(\bar{a})$ = false, else $P_{\varphi}(\bar{a})$ is truth (and appropriate "for all" also adding dummy variables is possible by "for all".

For conjunctions $\varphi(\bar{x}) = \varphi_1(\bar{x})$ use "if $P_{\varphi(\bar{x})}(\bar{a}) =$ truth then $P_{\varphi(\bar{x})}(\bar{a}) = P_{\varphi_2(\bar{x})}(\bar{a})$ else $P_{\varphi(\bar{x})}(\bar{a}) =$ false.

For existential quantifier, $\varphi(\bar{x}) = (\exists y)\psi(y, \bar{x})$ use
"if $P_{\psi(\bar{x},bary)}(\bar{a}) =$ truth then $P_{\varphi(\bar{x})}(\bar{a}) =$ truth else do nothing". $\square_{4.3}$

§5 Closing Comments

We may consider a context is $(K, \mathscr{I})$ such that logics related to our proof in §2. The first version in 5.1(3) changes the satisfaction the second (in 5.1(4),(4A)) changes also the syntax.

5.1 Definition. 1) A context is a pair $(K, \mathscr{I})$ such that

- (a) K be a class of models with vocabulary (= the set of predicates) τ
- (b) $\mathscr{I}$ is a function
- (c) $\text{Dom}(\mathscr{I}) = K$
- (d) $\mathscr{I}(M)$ is a family of subsets of M, whose union is $|M|$, and closed under subsets.

1A) We call $(K, \mathscr{I})$ invariant if

- (e) $\mathscr{I}$ is preserved by isomorphisms, i.e. if f is an isomorphism from $M_1 \in K$ onto $M_2 \in K$ and $A \in \mathscr{I}(M_1)$ then $f"(A) \in \mathscr{I}(M_2)$.

1B) We say "f is an $\mathscr{I}$-isomorphism from $M_1 \in K$ onto $M_2 \in K$" <u>if</u> f is an isomorphism from M_1 onto M_2 such that $\mathscr{I}(M_2) = \{f"(A) : A \in \mathscr{I}(M_1)\}$. Recall that $\mathscr{I}(M_2)[k] = \{\bigcup_{\ell<k} A_\ell : A_\ell \in \mathscr{I}[M]\}$.

2) In 1) let

$$\mathrm{Seq}^{\alpha}_{\mathscr{I}}(M) = \{\bar{a} : \bar{a} \text{ a sequence of members of } M \text{ of length } \alpha,\ \mathrm{Rang}(\bar{a}) \in \mathscr{I}(M)\}.$$

3) Let $\mathscr{L} = \mathscr{L}_{\mathrm{f.o.}}$ or $\mathscr{L} = \mathscr{L}_{\lambda,\kappa}$; recalling that the logic $\mathscr{L}_{\lambda,\kappa,\alpha}$ for λ, κ cardinals, is defined like first order logic but we allow conjunctions of $\bigwedge_{i<\alpha}$, for $\alpha < \lambda$ and existential quantifier $(\exists \bar{x})$ with $\bar{x}$ a sequence of variables of length $< \kappa$, and the depth of the formulas is $< \alpha$. We define $\mathscr{L}^{[k]} = \mathscr{L}^{[k]}_{\lambda,\kappa,\alpha}$, logics with the same syntax but with a difference in the definition of the satisfaction relation, $M \models^{[k]}_{\mathscr{I}} \varphi[\bar{a}]$ or $(M, \mathscr{I}) \models^{[k]} \varphi(\bar{a})$ is defined inductively on α as usual, except that

- $(*)$ we demand $\mathrm{Rang}(\bar{a}) \in \mathscr{I}[M][k]$ (otherwise not defined) , that is
  $M \models^{[k]}_{\mathscr{I}} (\exists \bar{x})\varphi(\bar{x}, \bar{b})$ <u>iff</u>
  $\mathrm{Rang}(\bar{b}) \in \mathscr{I}[M][k-1]$ and for some $\bar{a} \in {}^{\ell g(\bar{x})}M$ with
  $\mathrm{Rang}(\bar{a}) \in \mathscr{I}(M)$ we have $M \models^{k}_{\mathscr{I}} \varphi[\bar{a}, \bar{b}]$.

Let $\mathscr{L}_{\lambda,\kappa,\alpha;\mathrm{card}}, \mathscr{L}_{\lambda,\kappa;+\mathrm{na}}, \mathscr{L}_{\lambda,\kappa;\mathrm{card},\mathbf{T}}$ be defined similarly (so

$$M \models^{[k]}_{I} \exists^{!\mu} \bar{x} \varphi(\bar{x}, \bar{b}) \text{ iff } (\bar{b} \in {}^{\ell g(\bar{b}))}M),\ \mathrm{Rang}(\bar{b}) \in \mathscr{I}(M)[k-1]$$
$$\text{and } \mu = |\{a : \{a\} \in I \text{ and } M \models^{[k]}_{I} \varphi[a, \bar{b}]\}|.$$

Omitting α means some α.
If λ, κ, α are omitted they are $\aleph_0$ (so $L_{\lambda,\kappa,\alpha}$ is first order).
4) We define a logic $\mathscr{L}^{[k]}_{\lambda,\kappa,\alpha}$. Let us define the formulas in $\mathscr{L}^{[k]}_{\lambda,\kappa,\alpha}$ by induction on α, each formula φ has the form $\varphi(\bar{x}_0, \bar{x}_1, \ldots, \bar{x}_{k_1-1}), k_1 \le k$, where the $\bar{x}_\ell$'s are pairwise disjoint sequences of variables of length $< \kappa$ (so if $\kappa = \aleph_0$, finite sequences) and every variable appearing freely in φ appear in one of those sequences (so any formula is coupled with such $\langle \bar{x}_0, \ldots, \bar{x}_{k_1-1} \rangle$, probably some not actually appearing).

<u>$\alpha = 0$</u>: quantifier free formula; i.e. any Boolean combination of atomic ones (with the right variables, of course).

<u>$\alpha + 1$</u>: α non-limit $\varphi(\bar{x}_0, \ldots, \bar{x}_{k_1-1})$ is from $\mathscr{L}^{[k]}_{\lambda,\kappa,\alpha}$ or is a Boolean combination of formulas of the form $(\exists \bar{y})\psi(\bar{x}_{i_0}, \ldots, \bar{x}_{i_{k_2-2}}, \bar{y})$ where $k_2 \le k, \psi(\bar{x}_{i_0}, \ldots, \bar{x}_{i_{k_2-2}}, \bar{y}) \in \mathscr{L}^{[k]}_{\lambda,\kappa,\alpha}$.

<u>α limit</u>: $\mathscr{L}^{[k]}_{\lambda,\kappa,\alpha} = \bigcup_{\beta<\alpha} \mathscr{L}^{[k]}_{\lambda,\kappa,\beta}$.

$\alpha+1, \alpha$ limit: $\mathscr{L}^{[k]}_{\lambda,\kappa,\alpha+1}$ is the set of $\varphi \in \mathscr{L}^{[k]}_{\lambda,\kappa,\alpha}$ or φ a Boolean combinations of members of $\mathscr{L}^{*}_{\lambda,\kappa,\alpha}$ of the right variables.

Let $\mathscr{L}^{*}_{\lambda,\kappa} = \bigcup\limits_{\alpha} \mathscr{L}^{[k]}_{\lambda,\kappa,\alpha}$ and $\mathscr{L}^{[*]}_{\lambda,\kappa,\alpha} = \bigcup\limits_{k<\omega} \mathscr{L}^{[k]}_{\lambda,\kappa,\alpha}$ and $\mathscr{L}^{[k]}_{\alpha} = L^{[k]}_{\aleph_0,\aleph_0,\alpha}$ and $\mathscr{L}^{[k]}_{<\alpha} = \bigcup\limits_{\beta<\alpha} \mathscr{L}^{[k]}_{\beta}$ and $\mathscr{L}^{[k]} = \mathscr{L}^{[k]}_{\aleph_0,\aleph_0}$ and $L^{[*]} = \bigcup\limits_{k<\omega} L_k$.

4A) We now define a satisfaction relation $M \models \varphi(\bar{a}_0, \dots, \bar{a}_{k_1-1})$ where $k_1 \le k$ (depending on $\mathscr{I}$).

I.e. we define by induction on α, for $\varphi(\bar{x}_0, \dots, \bar{x}_{k_1-1}) \in \mathscr{L}^{[k]}_{\lambda,\kappa,\alpha}, \bar{a}_\ell \in \mathrm{Seq}^{\ell g(\bar{x}_\ell)}_{\mathscr{I}}(M)$, when does $M \models \varphi[\bar{a}_0, \dots, \bar{a}_{k_1-1}]$ and when $M \models \neg\varphi[\bar{a}_0, \dots, \bar{a}_{k_1-1}]$. This is done naturally, in particular $M \models (\exists \bar{y})\varphi(\bar{a}_0, \dots, \bar{a}_{k_2-2}, \bar{y})$ iff for some $\bar{b} \in \mathrm{Seq}^{\ell g(\bar{y})}_{\mathscr{I}}(M)$, (so $\mathrm{Rang}(\bar{b}) \in \mathscr{I}(M)$) we have $M \models \varphi[\bar{a}_0, \dots, \bar{a}_{k_2-2}, \bar{b}]$.

5) We can define for $\mathscr{L}$ one of the above, $(\mathscr{L})^{\mathrm{card}}$ similarly adding the quantifiers $[\varphi'(\bar{x};\bar{z})/\varphi''(\bar{x},\bar{y},\bar{z})]$ saying: $\varphi''(\dots,\dots;\bar{z})$ define an equivalence relation on $\{\bar{x} : \varphi'(\bar{x})\}$ with exactly s equivalence classes.

6) We can above replace models M by pairs $(M, I), I \subseteq \mathscr{P}(M)$ closed under subsets.

5.2 Discussion: We may replace M by M^+, adding elements coding each $A \in \mathscr{I}(M)$, with decoding by functions, but

- (a) this does not capture $\models^{[k]}$ and
- (b) for $\models^{[k]}$ this requires infinitely many functions, we need to actually code any sequence listing each $A \in \mathscr{I}(M)$.

Still this framework seems to work quite smoothly for its purposes. We could have made it more central (use 5.5 below).

Note that

5.3 Observation. For any k-system $\mathscr{Y}$ and $\mathfrak{Z}$ and $\mathbf{m}^{\Upsilon}_1$-lifting of $\mathscr{Y}$, letting $M^{\mathscr{Y}} = M$ we have

$$\mathscr{Y}_{\mathfrak{Z}} = (N^{\mathfrak{Z}}, \{A : (\exists f \in \mathscr{F})[\mathrm{Dom}(f) \in I^{\mathscr{F}} \ \&\ A \subseteq \mathrm{Dom}(G(f))]\}, \{G(f) : f \in \mathscr{F}\})$$

is a k-system.

Now easily (and it makes a connection with §1, §2):

5.4 Observation: 1) Let $(K, \mathscr{I})$ be a context and $M_1, M_2 \in K$ be (finite as always), τ-models, τ a vocabulary, and $k < \omega$.

The following are equivalent:

- (A) there are k-systems $\mathscr{Y}_\ell = (M_\ell, \mathscr{I}(M_\ell), \mathscr{F}_\ell)$ for $\ell = 1, 2$ and $\mathscr{H}$ as in Definition 2.21(1),(2)

(B) for every sentence $\psi \in \mathscr{L}^{[k]}_{\text{f.o.}}(\tau)$ we have
$M_1 \models^{[k]}_{\mathscr{J}} \psi \Leftrightarrow M_2 \models^{[k]}_{\mathscr{J}} \psi$

(C) for infinite λ, κ, α, for every sentence $\psi \in \mathscr{L}^{[k]}_{\lambda,\kappa,\alpha}(\tau)$ we have
$M_1 \models^{[k]}_{\mathscr{J}} \psi \Leftrightarrow M_2 \models^{[k]}_{\mathscr{J}} \psi$

(D) for every t and infinite λ, κ, α for every sentence $\psi \in \mathscr{L}^{[k]}_{\lambda,\kappa,\alpha}(\tau)$ we have
$N^{\mathfrak{Z}^{\text{full}}\mathscr{Y}_{1,t}} \models^{[k]}_{I_t^{\mathscr{Y}_1}} \psi \Leftrightarrow N^{\mathfrak{Z}^{\text{full}}\mathscr{Y}_{2,t}} \models^{[k]}_{I_t^{Y_2}} \psi$ where $N^{\mathfrak{Z}^{\text{full}}\mathscr{Y}_{\ell,t}}$ is defined in 2.13(7) and $I_t^{\mathscr{Y}} = \{\text{Dom}(f) : f \in G^{\mathfrak{Z}^{\text{full}}\mathscr{Y}_{,t}}\}$.

$\square_{5.4}$

Proof. Straight.

5.5 Definition. 1) We say Υ (from Definition 1.1) is pure if $m_1[\Upsilon] = 0$ so no P_ℓ.
2) Let $\mathscr{Y} = (M, I, \mathscr{F})$ be a k-system; let "$\mathfrak{Z}^*$ is the full $\mathbf{t}$-successor of $\mathfrak{Z}$" be as defined in 2.13. We define by induction on t; $\mathfrak{Z}_t = \mathfrak{Z}^t[\mathscr{Y}, \mathbf{t}]$ as follows: for $t = 0, N^{\mathfrak{Z}^t} = M, P_\ell^{\mathfrak{Z}^t}$ is the empty set, $G^{\mathfrak{Z}^t}$ is the identity on $\mathscr{F}$ and $R^{\mathfrak{Z}} = \{(A, x) : A \in I, x \in A\}$; for $t = s + 1$ let $\mathfrak{Z}^t$ be the full $\mathbf{t}$-successor of $\mathfrak{Z}^s$. Let $\mathscr{I}^{\mathfrak{Z}} = I_{\mathfrak{Z},\mathscr{Y}} = \{B : B \subseteq S_{\mathfrak{Z}}(A)$ for some $A \in I\}$ where $S_{\mathfrak{Z}}(A) = S_{\mathfrak{Z},A} = \{x \in N^{\mathfrak{Z}} : A$ is a $\mathfrak{Z}$-support of $x\}$.

We can also see:

5.6 Claim. *1) Assume*

(a) $\mathscr{Y}_\ell = (M_\ell, \mathscr{I}_\ell, \mathscr{F}_\ell)$ *is a* $\mathbf{t}$*-dichotomical* k*-system for* $\ell = 1, 2$
(b) $\mathfrak{Z}^\ell_t = \mathfrak{Z}_t[M_\ell, I_\ell]$, *so* $\mathfrak{Z}^\ell_{t+1}$ *is the full successor of* $\mathfrak{Z}^\ell_t$.

Then the following are equivalent:

(α) $(M_1, I_1), (M_2, I_2)$ *are* $\mathscr{L}^{[k]}$*-equivalent*
(β) *for every* $t < \infty$ *the pairs* $(M_1^{\mathfrak{Z}_t}, I^{\mathfrak{Z}_t}), (M_2^{\mathfrak{Z}_t}, I^{\mathfrak{Z}_2})$ *are* $\mathscr{L}^{[k]}$*-equivalent.*

2) For any given (M, I) *there in a sentence* $\psi \in \mathscr{L}^{[k]}_{\text{f.o.}}(\tau_M)$ *satisfied by* (M, I) *and implying any other sentence* $\psi' \in L^{[k]}_{\infty,\infty}(\tau_\mu)$ *satisfied by* (M, I).

5.7 Remark. 1) So in part (2) we can apply 2.22, 2.23, 2.24.
2) Note that by 5.3, $\mathscr{L}^{[k]}$ satisfies addition theorems.

5.8 Fact: For any Υ we can find Υ' which is equivalent if we use in Definition 1.3 the case $\iota = 4$ (well when $\mathbf{t}(\|M_\ell\|)$ always is ≥ 2). In fact, we can reconstruct (i.e. define by a formula in $\mathscr{L}^*$) the sequence of $\langle P_{t,\ell} : t' < t\rangle$ in N_t.

5.9 Conclusion 1) Assume $\mathscr{Y}$ is counting k-system (see 2.3). Then we can define R_t, G_t for every t (N_t the "computation" in time t) such that

$$(M, \bar{P}_0, G_0, R_0) \text{ is 0-lifting}$$

$$(N_{t+1}, \bar{P}_{t+1}, G_{t+1}, R_{t+1}) \text{ is a lifting, successor of } (N_t, \bar{c}_t, G_t, R_t).$$

2) So the formula the $\bar{\varphi}$ defines is preserved by $f \in \mathscr{F}_0$.

Proof. Straight.

* * *

5.10 Discussion: 1) In §2 and in 5.5 we can allow infinite models M and define $N_t[M] = N_t[M, \Upsilon, \mathbf{t}]$ for every ordinal t, for this better assume Υ is standard, monotonic and pure (or strongly monotonic, i.e. demand $P_{\ell,t}[M, \Upsilon, \mathbf{t}]$ is increasing with t; for t limit we take union and so $V[M, \Upsilon, \mathbf{t}] = \cup\{N_\alpha[M, \Upsilon, \mathbf{t}] : \alpha \text{ an ordinal}\}$, see below. Now as in the case $\iota = 4$, the analysis in §2 works for this but it is not clear if we can get any interesting things.
Note that those definitions remind us of Gödel's construction of L, particularly of $L_{\alpha+1}$ from L_α, and the Frankel-Mostowski models (which use automorphisms).

May can this give interesting proofs of consistency for set theory with no choice but with urelements? It seems they can be reduced to the classical case.
2) We can prove various equivalence and 0-1 laws by 5.5, by proving that the relevant model N_t can be interpreted in $\mathfrak{Z}^t[M, I]$ from 5.5, using f.o. logic which suffices.

Proof. Straight.

References

[References of the form `math.XX/···` refer to the `xxx.lanl.gov` archive]

[BlSh 528] John Baldwin and Saharon Shelah. Randomness and Semigenericity. *Transactions of the American Mathematical Society*, **349**:1359–1376, 1997. `math.LO/9607226`

[BGSh 533] Andreas Blass, Yuri Gurevich, and Saharon Shelah. Choiceless Polynomial Time. *Annals of Pure and Applied Logic*, **100**:141–187, 1999. `math.LO/9705225`

[CK] Chen C. Chang and Jerome H. Keisler. *Model Theory*, volume 73 of *Studies in Logic and the Foundation of Math.* North Holland Publishing Co., Amsterdam, 1973.

[Di] M. A. Dickman. Larger infinitary languages. In J. Barwise and S. Feferman, editors, *Model Theoretic Logics*, Perspectives in Mathematical Logic, chapter IX, pages 317–364. Springer-Verlag, New York Berlin Heidelberg Tokyo, 1985.

[EbFl95] Heinz-Dieter Ebbinghaus and Jorg Flum. *Finite model theory.* Perspectives in Mathematical Logic. Springer-Verlag, Berlin, 1995.

[GuSh 526] Yuri Gurevich and Saharon Shelah. On finite rigid structures. *Journal of Symbolic Logic*, **61**:549–562, 1996. `math.LO/9411236`

[Ho93] Wilfrid Hodges. *Model theory*, volume 42 of *Encyclopedia of Mathematics and its Applications.* Cambridge University Press, Cambridge, 1993.

[Ka] Carol R. Karp. Finite quantifier equivalence. In J. W. Addison, L.A. Henkin, and A. Tarski, editors, *The Theory of Models*, pages 407–412. North Holland Publ. Co, 1965.

[Sh 550] Saharon Shelah. 0–1 laws. *Preprint.* `math.LO/9804154`

[Sh:a] Saharon Shelah. *Classification theory and the number of nonisomorphic models*, volume 92 of *Studies in Logic and the Foundations of Mathematics.* North-Holland Publishing Co., Amsterdam-New York, xvi+544 pp, $62.25, 1978.

[Sh:c] Saharon Shelah. *Classification theory and the number of nonisomorphic models*, volume 92 of *Studies in Logic and the Foundations of Mathematics.* North-Holland Publishing Co., Amsterdam, xxxiv+705 pp, 1990.

Schema Revisited

Colin Stirling

Division of Informatics
University of Edinburgh
Edinburgh EH9 3JZ, UK
cps@dcs.ed.ac.uk

1 Introduction

Two schema problems from the 1970s are examined, monadic recursion schemes and first-order recursion schemas. Research on these problems halted because they were shown to be equivalent to the problem of decidability of language equivalence between DPDA (deterministic pushdown automata). Recently a decidability proof for equivalence of DPDA was given by Sénizergues [10,11], which therefore also solves the schema problems. However Sénizergues proof is quite formidable. A simplification of the proof was presented by the author [13] using ideas from concurrency theory (for showing decidability of bismilarity [9,12]) and crucial insights from Sénizergues's intricate proof.

In this abstract we concentrate on first-order schemes and we outline a solution, based on the DPDA equivalence proof, which is reasonably close to its original formulation. We make use of Courcelle's work [1,2], which shows how to reduce this schema problem to decidability of language equivalence between strict deterministic grammars. And the proof in [13] of decidability of DPDA equivalence proceeds via (a small extension of) these grammars.

2 Monadic Recursion Schemes

A monadic recursion scheme, following Garland and Luckham [6], is defined relative to a set of basis monadic functions $\mathsf{F} = \{f_1, \ldots, f_k\}$ and a set of predicates $\mathsf{P} = \{P_1, \ldots, P_l\}$ as a finite family

$$\begin{array}{l} F_1 x \stackrel{\text{def}}{=} \text{if } P_1 x \text{ then } \alpha_1 x \text{ else } \beta_1 x \\ \vdots \qquad\qquad \vdots \\ F_n x \stackrel{\text{def}}{=} \text{if } P_n x \text{ then } \alpha_n x \text{ else } \beta_n x \end{array}$$

where each F_i is distinct and each α_i and β_i is a string of defined and basis functions, a member of $(\mathsf{F} \cup \mathsf{DF})^*$ when $\mathsf{DF} = \{F_1, \ldots, F_n\}$, and each $P_i \in \mathsf{P}$. The scheme is usually identified by the initial defined function F_1.

Example 1 A simple example is $Fx \stackrel{\text{def}}{=} \text{if } Px \text{ then } fx \text{ else } FFfx$. □

P. Clote and H. Schwichtenberg (Eds.): CSL 2000, LNCS 1862, pp. 126–138, 2000.

A computation of a scheme is defined with respect to an interpretation I which fixes the meanings of the basis functions, predicates and variable. An interpretation I over a non-empty value space D is a mapping such that $I(x) \in D$, $I(f_i) \in D \to D$ and $I(P_i) \in D \to \{\mathtt{tt}, \mathtt{ff}\}$. A computation relative to I is then defined using the following transition rules

$$F_1 x \to F_1 d \quad \text{if} \quad I(x) = d \qquad\qquad \delta f_i d \to \delta d' \quad \text{if} \quad I(f_i)(d) = d'$$
$$\delta F_i d \to \delta \alpha_i d \text{ if } \quad I(P_i)(d) = \mathtt{tt} \qquad \delta F_i d \to \delta \beta_i d \text{ if } \quad I(P_i)(d) = \mathtt{ff}$$

The value of a scheme Fx relative to I, written $\mathrm{Val}_I(Fx)$, is a member of $D_\perp$. If $Fx \longrightarrow^* d$ then $\mathrm{Val}_I(Fx) = d$ and if the computation never ends $\mathrm{Val}_I(Fx) = \perp$.

Two schemes F and G are strongly equivalent, written $F \sim G$, if for all interpretations I, $\mathrm{Val}_I(Fx) = \mathrm{Val}_I(Gx)$. The classical equivalence problem for monadic recursion schemes is to show whether or not there is a decision procedure for $F \sim G$. For background and significance of the problem, see Garland and Luckham [6] and references cited therein.

An interpretation I is free if the domain $D = \mathsf{F}^*\{x\}$ and $I(f) = f$ and $I(x) = x$. The result of a computation relative to a free I is a word (or $\perp$).

Example 2 Let I be the free interpretation for example 1 where $I(P)(f^n x) = \mathtt{tt}$ iff $n > 0$. Therefore $Fx \longrightarrow FFfx \longrightarrow Fffx \longrightarrow fffx$, and $\mathrm{Val}_I(Fx) = fffx$. In contrast, if $I(P)(f^n x) = \mathtt{tt}$ iff n is odd then $\mathrm{Val}_I(Fx) = \perp$. □

Relationships with language theory were underpinned by the following result in Garland and Luckham.

Fact 1 *$F \sim G$ iff for all free interpretations I, $\mathrm{Val}_I(Fx) = \mathrm{Val}_I(Gx)$.*

Garland and Luckham showed that the decision problem for schemes reduces to the problem of decidability of DPDA, and Friedman showed that the converse also holds using jump DPDA [5]. We shall now provide a cleaner variant reduction to the DPDA problem, inspired by these authors.

First we assume a "Greibach" normal form for a scheme. In $F_i \stackrel{\text{def}}{=}$ if $P_i x$ then $\alpha_i x$ else $\beta_i x$ each α_i and β_i has one of the forms ϵ or f_j or F_j or $F_j f_k$ or $F_j F_k$. It is straightforward to transform a scheme into normal form by adding auxilary definitions. For instance, example 1 becomes $Fx \stackrel{\text{def}}{=}$ if Px then fx else Gfx and $Gx \stackrel{\text{def}}{=}$ if Px then FFx else FFx.

Let B be the set of boolean arrays of size l. If $b \in \mathsf{B}$ then b_i is the ith entry of b: the idea is that $b_i = \mathtt{tt}$ means that P_i is true. A stack symbol is an element of DF, a state is a boolean b and the alphabet consists of elements bfb' where $f \in \mathsf{F}$. A configuration of the DPDA has the form $b\delta$ where $\delta \in \mathsf{DF}^*$ is a sequence of stack symbols. Transitions have the form $b\delta \xrightarrow{bfb'} b'\delta'$ or $b\delta \xrightarrow{\epsilon} b'\delta'$. Let δ^R be the reverse of δ. Assume a scheme with definition $Fx \stackrel{\text{def}}{=}$ if $P_i x$ then αx else βx.

Basic transitions for bF are determined from the scheme as follows.

$$\begin{array}{rl} bF \xrightarrow{bfb'} b'\alpha' & \text{if either } b_i = \mathtt{tt} \text{ and } \alpha^R = f\alpha' \\ & \text{or} \quad b_i = \mathtt{ff} \text{ and } \beta^R = f\alpha' \\ bF \xrightarrow{\epsilon} b\alpha' & \text{if either } b_i = \mathtt{tt} \text{ and } \alpha^R = \alpha' \text{ and } \alpha \in \mathsf{DF}^* \\ & \text{or} \quad b_i = \mathtt{ff} \text{ and } \beta^R = \alpha' \text{ and } \beta \in \mathsf{DF}^* \end{array}$$

There is also the prefix rule, if $bF \xrightarrow{a} b'\alpha'$ then $bF\delta \xrightarrow{a} b'\alpha'\delta$. Basic transitions obey the following two deterministic properties:

$$\begin{array}{l} \text{if } bF \xrightarrow{\epsilon} b\delta \text{ then not}(bF \xrightarrow{bfb'} b'\delta') \text{ for any } b'\delta' \\ \text{if } bF \xrightarrow{a} b'\delta' \text{ and } bF \xrightarrow{a} b''\delta'' \text{ then } b' = b'' \text{ and } \delta' = \delta'' \end{array}$$

The result is therefore a DPDA.

The language of a configuration $b\delta$, written $L(b\delta)$, is the set of words $w \in (\mathsf{B} \times \mathsf{F} \times \mathsf{B})^*$ recognised by $b\delta$ using these transition rules, where ϵ-transitions are swallowed in the usual way, and assuming empty stack acceptance, $L(b\delta) = \{w : b\delta \xrightarrow{w} b'\epsilon \text{ for some } b'\}$. The following is a consequence of Fact 1 and the construction.

Proposition 1 $F \sim G$ iff for all b, $L(bF) = L(bG)$.

There is a routine transformation of a DPDA into a context-free grammar (which is strict deterministic [7,8], more on this later). First one transforms the DPDA into normal form where ϵ-transitions only pop the stack, by examining what happens under repeated basic ϵ-transitions. Next one transforms the normalised DPDA into a context free grammar whose nonterminals are triples of the form bFb' and whose alphabet is the same as that of the DPDA. The idea is that, for instance, a basic transition of the DPDA of the form $bF \xrightarrow{a} b''GH$ becomes in the grammar the family of transitions for each b''', $bFb' \xrightarrow{a} b''Gb'''b'''Hb'$. Hence the language accepted by the nonterminal bFb', $L(bFb')$, is the set of words w such that $bF \xrightarrow{w} b'\epsilon$. Hence $F \sim G$ iff for all b and b', $L(bFb') = L(bGb')$. Decidability of monadic recursion schemes follows from the following theorem.

Theorem 1 *It is decidable whether* $L(bFb') = L(bGb')$.

In this abstract there is only an intimatation of the procedure in section 4, because we shall concentrate on the second schema problem.

3 Recursive Program Schemes

A recursive program scheme, following Courcelle [1,2], is defined relative to a set of basis functions $\mathsf{F} = \{f_1, \ldots, f_k\}$ and a set of basis variables $\mathsf{V} = \{x_1, \ldots, x_l\}$. Each basis function f has an associated arity $\rho(f) > 0$, and therefore need not

be monadic. A scheme is a finite family of the form

$$\begin{aligned} F_1(x_1,\ldots,x_{m_1}) &\stackrel{\text{def}}{=} t_1 \\ \vdots \qquad\qquad\qquad &\quad\ \vdots \\ F_n(x_1,\ldots,x_{m_n}) &\stackrel{\text{def}}{=} t_n \end{aligned}$$

where each F_i is distinct, and has an associated arity $\rho(F_i) = m_i$, and where terms t_i are built from the basis and defined functions and variables, and therefore have the form x_j or $f_j(t_1,\ldots,t_{\rho(f_j)})$ or $F_j(t_1,\ldots,t_{\rho(F_j)})$. A scheme is again usually identified with its head function F_1. We let DF be the set of defined functions.

Example 1 A simple example is $F(x) \stackrel{\text{def}}{=} f(F(gx), g(x))$.

The interpretation of a scheme is either the undefined tree or a completed tree whose depth is finite or infinite and where internal nodes are labelled with elements of F and leaves are labelled with elements of V. The following transition rules generate the tree and they are applied down the depth of the tree starting with $F_1(x_1,\ldots,x_{\rho(F_1)})$

$$\begin{array}{l} x_i \longrightarrow x_i \\ F_i(t'_1,\ldots,t'_{\rho(F_i)}) \longrightarrow t_i\{t'_1/x_1,\ldots,t'_{\rho(F_i)}/x_{\rho(F_i)}\} \\ \text{if } t'_j \longrightarrow t''_j,\ 1 \le j \le \rho(f_i),\ \text{then } f_i(t'_1,\ldots,t'_{\rho(f_i)}) \longrightarrow f_i(t''_1,\ldots,t''_{\rho(f_i)}) \end{array}$$

where $\{\cdot/\cdot\}$ is simultaneous substitution. For instance in the case of example 1

$$Fx \longrightarrow f(F(gx),gx) \longrightarrow f(f(F(ggx),ggx),gx) \longrightarrow \ldots$$

The value of a scheme belongs to the family $T^{\omega}_{\perp}$ of appropriate trees.

Alternatively one can view $T^{\omega}_{\perp}$ as a domain[1] of trees. The meaning of a scheme is the least fixed point with respect to the free tree interpretation, following Damm [3,4]. In the case of example 1

$$F^0(x) = \perp \qquad\qquad F^{i+1}(x) = f(F^i(gx), gx)$$

So $F^2(x) = f(((f\ \perp), ggx), gx)$. The resulting tree in $T^{\omega}_{\perp}$ is $F^{\omega}(x) = \bigsqcup_{i\ge 0} F^i(x)$ which is the meaning of $Y(\lambda F.\,\lambda x.\, f(F(gx),gx))(x)$ with respect to the free interpretation. Thus schemes are only "first-order". Higher order schemes are considered by Damm [3,4].

Two schemes F and G with arity n are equivalent, written $F \sim G$, if they produce the same tree, that is if $F^{\omega}(x_1,\ldots,x_n) = G^{\omega}(x_1,\ldots,x_n)$. The classical equivalence problem for recursion schemes is to show whether or not there is a decision procedure for $F \sim G$. For background and significance of the problem see [1,2] and references cited therein.

[1] With ordering $\perp \sqsubseteq T$ and $T_i \sqsubseteq T'_i$ for each i implies $f_j(T_1,\ldots,T_{\rho(f_j)}) \sqsubseteq f_j(T'_1,\ldots,T'_{\rho(f_j)})$.

The equivalence problem for schemes was shown to be interreducible to the DPDA problem by Courcelle [2], via grammars. A key idea is to represent a tree $T \in T^{\omega}_{\perp}$ as the language of its finite branches, $B(T)$. The following finite tree $f(g(x_1, x_2), f(x_1, h(x_3)))$ is given as $\{f^1g^1x_1, f^1g^2x_2, f^2f^1x_1, f^2f^2h^1x_3\}$. Each word is a branch. One splits each basis function f of arity k into terminal symbols $f^1, \ldots, f^k$ reflecting the different directions that can be taken to obtain the branch. In the case of T generated by example 1 above $B(T)$ is the deterministic context-free language $\{f^2g^1x, f^1f^2g^1g^1x, f^1f^1f^2g^1g^1g^1x, \ldots\}$.

If $T = T'$ then $B(T) = B(T')$. But the converse need not hold. Consider the trees generated by the schemes $Fx \stackrel{\text{def}}{=} f(Fx)$ and $Gx \stackrel{\text{def}}{=} g(Gx)$. These trees are not "locally finite" [2]. A tree T is locally finite if whenever u is a prefix of a branch of T then there is a finite word v such that $uv \in B(T)$. Locally finite trees with the same branch language are equal. It is straightforward as Courcelle notes to guarantee local finiteness by increasing the arity of the basis functions by one and adding a new variable. Consider the transformed schemes $Fxy \stackrel{\text{def}}{=} f(Fxy, y)$ and $Gxy \stackrel{\text{def}}{=} g(Gxy, y)$. Two schemes are equivalent iff their transformations are also equivalent. Hence we can restrict attention to schemes that generate locally finite trees, and for these the following holds, as shown by Courcelle [2].

Fact 1 *$F \sim G$ iff $B(F^{\omega}(x_1, \ldots, x_n)) = B(G^{\omega}(x_1, \ldots, x_n))$.*

Following Courcelle, the next step is to transform a scheme into a context-free grammar which generates its branch language. We exclude the case where a scheme generates a single node tree x_i: it is easy to directly check equivalence betweeen such schemes. With this exclusion, we assume that schemes are given in "Greibach" normal form. Each term t_i in the definition of F_i has the form $f(rt_1, \ldots, rt_{\rho(f)})$ where $f \in \mathsf{F}$ and where each rt_j is built from variables and defined function symbols only and where the depth of their embedding is at most two. This normal form is easy to achieve by introducing auxilary defined functions.

An ϵ-free context-free grammar in 3-Greibach normal form consists of a finite set N of nonterminals, a finite alphabet A and a finite family of basic transitions, each of the form $X \stackrel{a}{\longrightarrow} \alpha$ where $X \in \mathsf{N}$, $a \in \mathsf{A}$ and $\alpha \in \mathsf{N}^*$ such that its length, $|\alpha|$ is less than 3. A simple configuration is a sequence of nonterminals whose behaviour is determined by the basic transitions and the prefix rule: if $X \stackrel{a}{\longrightarrow} \alpha$ then $X\beta \stackrel{a}{\longrightarrow} \alpha\beta$ where $\beta \in \mathsf{N}^*$. The language accepted by a simple configuration α, $L(\alpha)$, is the set of words $\{w \in \mathsf{A}^* : \alpha \stackrel{w}{\longrightarrow} \epsilon\}$.

Given a scheme in normal form we associate a grammar with it as follows. The alphabet A is the set of split functions f^j, $1 \leq j \leq \rho(f)$ for $f \in \mathsf{F}$. The nonterminals N is the set F^j, $1 \leq j \leq \rho(F)$. Assume in the scheme that $F(x_1, \ldots, x_{\rho(F)}) \stackrel{\text{def}}{=} f(rt_1, \ldots, rt_{\rho(f)})$. The basic transitions are defined as fol-

lows, for each nonterminal F^i and alphabet symbol f^j.

$$\begin{aligned}
F^i &\xrightarrow{f^j} \epsilon \qquad \text{if } rt_j = x_i \\
F^i &\xrightarrow{f^j} G^k \qquad \text{if } rt_j = G(rt'_1, \ldots, rt'_{\rho(G)}) \text{ and } rt'_k = x_i \\
F^i &\xrightarrow{f^j} G^k H^l \text{ if } rt_j = G(rt'_1, \ldots, rt'_{\rho(G)}) \text{ and} \\
&\qquad\qquad\qquad\quad rt'_k = H(rt''_1, \ldots, rt''_{\rho(H)}) \text{ and } rt''_l = x_i
\end{aligned}$$

The grammar is defined so that the language accepted by a nonterminal F^i is the set of words $w \in \mathsf{A}^+$ such that $wx_i \in B(F^\omega(x_1, \ldots, x_{\rho(F)}))$. For example in the case of the second transition rule because G has x_i in its kth position in the definition of F it follows that $\{f^j w : w \in L(G^k)\} \subseteq L(F^i)$. Hence the following result holds.

Fact 2 *$F \sim G$ iff for each i, $L(F^i) = L(G^i)$.*

4 The Decision Procedure

The disjoint union of two recursion schemes is a single scheme and therefore we need only consider a single grammar. The equivalence problem is then to show that for each i, $L(F^i) = L(G^i)$ for F, $G \in \mathsf{DF}$. We assume that the grammar is "tidied" as usual by removing redundant nonterminals (those not reachable from any F^i and G^i and those whose language is $\emptyset$). In the following we use X, Y and Z to range over nonterminals and α, β to range over sequences of nonterminals.

The decision procedure consists of two semi-decision procedures. One half is easy, if $L(F^i) \neq L(G^i)$ then there is a smallest word which distinguishes them. The other half is more difficult. We show that $F \sim G$ iff there is a finite tableau proof for this. Tableaux have been used for proving decidability of bisimulation equivalence [9,12]. They are also implicit in Sénizergues's proof [11] where they appear as strategies.

The tableau proof system is goal directed, and consists of two kinds of rules, simple and conditional. Simple rules have the form

$$\frac{\text{Goal}}{\text{Subgoal}_1 \ \ldots \ \text{Subgoal}_n}\ \mathcal{C}$$

where Goal is what currently is to be proved and the subgoals are what it reduces to, provided the side condition $\mathcal{C}$ holds. A conditional rule has the form

$$\frac{\begin{array}{c}\text{Goal}_1 \\ \vdots \\ \text{Goal}_k \\ \vdots \\ \text{Goal}\end{array}\quad \mathcal{C}}{\text{Subgoal}}$$

where Goal is the current goal to be shown and there is a single subgoal to which it reduces provided that the goals $\text{Goal}_1,\ldots,\text{Goal}_k$ occur above Goal on the path between it and the root (starting goal) and provided that the side condition $\mathcal{C}$ holds. The use of conditional tableau rules is a new innovation, which is essentially due to Sénizergues.

There is also the important notion of when a current goal counts as final. Final goals are classified as either successful or unsuccessful. A *tableau proof* for a starting Goal is a finite proof tree, whose root is Goal and all of whose leaves are successful final goals, and all of whose inner subgoals are the result of an application of one of the rules.

The first tableau proof rule is the initial simple rule, INIT.

$$\frac{F = G}{F^1 = G^1 \ldots F^n = G^n}$$

The initial goal $F = G$, "are schemes F and G equivalent?" reduces to the subgoals $F^i = G^i$, "is $L(F^i) = L(G^i)$?", for each i.

The main idea of the tableau proof system is to reduce goals to subgoals by following branches down the trees for F and G. F^i represents the subtree for F all of whose branches end in x_i. The configuration $(F^i \cdot w)$ represents the subtree for F given by taking path w down the subtree F^i: it is therefore the subtree whoses branches are $\{v : wvx_i \text{ is a branch of the tree for } F\}$. Clearly $F \sim G$ iff for all w and i, $(F^i \cdot w) \sim (G^i \cdot w)$. We show that the subtree $(F^i \cdot w)$ is naturally described in the grammar.

Basic transitions of the grammar induced by a scheme are "almost" deterministic. If $F^i \xrightarrow{f^j} \epsilon$ and $F^i \xrightarrow{f^j} \alpha$ then $\alpha = \epsilon$ because x_i is in the jth position of f and nothing else is thereby allowed. However if the jth position of f is $G(rt'_1, \ldots, rt'_{\rho(G)})$ then it is possible that $F^i \xrightarrow{f^j} \alpha$ and $F^i \xrightarrow{f^j} \beta$ when $\alpha \neq \beta$. However α and β are "similar": if $\alpha = G^k\alpha'$ then β must have the form $G^l\beta'$ and if $l = k$ then α' and β' must again be similar (both of the form $H^{l'}$). The grammar is in fact strict deterministic [7,8].

Let $\equiv$ be a partition of the nonterminals N of a context-free grammar (in normal form). The partition $\equiv$ is extended to sequences of nonterminals, $\alpha \equiv \beta$ if $\alpha = \beta$ or there is a δ such that $\alpha = \delta X\alpha_1$ and $\beta = \delta Y\beta_1$ and $X \equiv Y$ and $X \neq Y$. A partition $\equiv$ on N is strict if the basic transitions obey the following two conditions:

$$\text{if } X \xrightarrow{a} \alpha \text{ and } Y \xrightarrow{a} \delta \text{ and } X \equiv Y \text{ then } \alpha \equiv \delta$$
$$\text{if } X \xrightarrow{a} \alpha \text{ and } Y \xrightarrow{a} \alpha \text{ and } X \equiv Y \text{ then } X = Y$$

A context-free grammar is strict deterministic if there exists a strict partition of its nonterminals. The partition on the grammar induced by a scheme is given by $F^i \equiv F^j$ for each F and indices i and j. Clearly the two strictness conditions hold[2]. Hence for $\alpha, \beta \neq \epsilon$ and $\alpha \neq \beta$, $\alpha \equiv \beta$ if $\alpha = \delta F^i\alpha'$ and $\beta = \delta F^j\beta'$ for $i \neq j$.

[2] Similarly the context-free grammar induced by a monadic recursion scheme is strict deterministic when the partition is given by $bFb' \equiv bFb''$

The strictness conditions generalise to words (replacing a with $w \in \mathsf{A}^+$ throughout). It therefore follows that if $X \equiv Y$ then the languages accepted by X and Y are prefix-disjoint and if $X \neq Y$ then they accept disjoint languages. That is, if $w \in L(F^i)$ and $i \neq j$ then no prefix of w including w belongs to $L(F^j)$. This is clear from the tree generated by F: if wx_i is a branch then this excludes vx_j as a branch whenever v is a prefix of w.

A simple configuration of a grammar is a sequence of nonterminals β. A composite configuration is a finite family of simple configurations, $\beta_1 + \ldots + \beta_n$. The language accepted by a composite configuration is the union of the languages acccepted by the components, $L(\beta_1 + \ldots + \beta_n) = \bigcup L(\beta_i)$. For simplicity we also assume that the empty sum, $\emptyset$, is also a configuration. Our main concern is with a subset of such configurations: $\beta_1 + \ldots + \beta_n$ is *admissible* if $\beta_i \equiv \beta_j$ for each pair of components β_i and β_j. Note that the singleton member ϵ is admissible and so is $\emptyset$. Subtrees of (the tree for) F such as $(F^i \cdot w)$ are represented as admissible configurations. Let $(F^i \cdot a)$ be defined as $\sum\{\alpha : F^i \xrightarrow{a} \alpha \text{ is a basic transition}\}$ which is an admissible configuration because the grammar is strict. $L((F^i \cdot a))$ is $\{w : awx_i \text{ is a branch in the tree for } F\}$. If $A = X_1\beta_1 + \ldots + X_n\beta_n$ is admissible then $(A \cdot a)$ is $\sum\{\alpha_{i_1}\beta_1 : X_1 \xrightarrow{a} \alpha_{i_1}\} + \ldots + \sum\{\alpha_{i_n}\beta_n : X_n \xrightarrow{a} \alpha_{i_n}\}$, which is also admissible. The notation is extended to words. $(A \cdot \epsilon) = A$ and $(A \cdot aw) = (A \cdot a) \cdot w$, where $(\emptyset \cdot w) = \emptyset$. It is easy to check that for any w, if A is admissible then $(A \cdot w)$ is also admissible.

We now return to the tableau construction. We let A, B, C and D range over admissible configurations. Goals in the tableau proof system (except for the initial goal $F = G$) have the form $A = B$. The next tableau proof rule is again a simple rule, UNF (unfold). Let $\mathsf{A} = \{a_1, \ldots, a_k\}$.

$$\frac{A = B}{(A \cdot a_1) = (B \cdot a_1) \quad \ldots \quad (A \cdot a_k) = (B \cdot a_k)}$$

UNF allows one to walk down the trees for F^i and G^i. UNF is the strategy T_A in Sénizuergues's proof.

The size of an admissible configuration $A = \beta_1 + \ldots + \beta_n$, written $|A|$, is the length of its largest sequence, $\max\{|\beta_i| : 1 \leq i \leq n\}$. A has many different "shapes", as it can be written in many different ways using obvious equalities (such as $B(C + D) = BC + BD$). A basic shape is a head nonterminal form $X_1A_1 + \ldots + X_kA_k$ where $X_i \neq X_j$, $i \neq j$, and $X_i \equiv X_j$. In this case the X_is are heads and A_is are tails. Another head form is $\alpha_1A_1 + \ldots + \alpha_nA_n + B$ where $\alpha_i \equiv \alpha_j$ and $|\alpha_i| = |\alpha_j|$ and no $A_j = \epsilon$ and $|B| \leq |\alpha_i|$. Instead one may focus on tail forms. If $(X_i \cdot w) = D_i$ (where D_i may be $\emptyset$ and for no prefix v of w is $X_i \cdot v = \epsilon$) then $(X_1A_1 + \ldots + X_kA_k \cdot w) = D_1A_1 + \ldots + D_kA_k$. The shape $D_1A_1 + \ldots + D_kA_k$ highlights the tails A_i. Because the grammar is in 3-Greibach normal form $|D_i| \leq 1 + |w|$ for each i.

Associated with any nonterminal F^i is a smallest word $w(F^i)$ such that $w(F^i) \in L(F^i)$, and so $(F^i \cdot w(F^i)) = \epsilon$. Note that if $(F^i \cdot v) = \epsilon$ and $j \neq i$ then $(F^j \cdot v) = \emptyset$. An important measure is M which is $\max\{|w(X)| : X \text{ is a nonterminal}\}$.

UNF allows one to proceed down the trees for F and G. Any subgoal $A = B$ can be thought of as $(F^i \cdot w) = (G^i \cdot w)$ where w is a prefix of a branch. The next step is to permit tree surgery and transplantation to "balance" the subtree expressions. We give the tableau rule BAL(L). This is a conditional tableau rule. In Sénizuergues's proof this is the strategy T_B.

$$\frac{\begin{array}{c} X_1A_1 + \ldots + X_kA_k = B \\ \vdots \\ D_1A_1 + \ldots + D_kA_k = B' \end{array} \quad \mathcal{C}}{D_1(B \cdot w(X_1)) + \ldots + D_k(B \cdot w(X_k)) = B'}$$

where $\mathcal{C}$ is the condition: there are exactly M applications of UNF between the top goal and bottom goal and no other rule is applied, and each $D_i \neq \epsilon$. To understand the rule assume that $D_1A_1 + \ldots + D_kA_k = B'$ is the current goal. This reduces to the subgoal beneath it provided that the top goal appears above it in the proof tree and condition $\mathcal{C}$ holds. There is also the symmetric rule BAL(R) where the premises are $B = \ldots$ and $B' = \ldots$, and the conclusion is $B' = \ldots$.

Consider the top goal of BAL(L), $A = B$. Let B have shape $\beta_1B_1 + \ldots + \beta_nB_n + C$ where $|\beta_i| = \text{M}+1$. Because $(X_i \cdot w(X_i)) = \epsilon$ it follows that $(A \cdot w(X_i)) = A_i$. Therefore if the top goal is true then $L(A_i) = L(B \cdot w(X_i))$. It is this substitution of $(B \cdot w(X_i))$ for A_i for each i in the bottom goal which the rule sanctions. Moreover $(B \cdot w(X_i))$ is $(\beta_1 \cdot w(X_i))B_1 + \ldots + (\beta_n \cdot w(X_i))B_n + (C \cdot w(X_i))$ because $|w(X_i)| < |\beta_j|$. Also B' has the shape $B'_1B_1 + \ldots + B'_nB_n + C'$ (where $|C'|, |B'_i| \leq 2\text{M} + 1$). Putting all this together it means that the subgoal has the following form, where some of the A'_i and B'_j may be $\emptyset$ and $B_{n+1} = \epsilon$.

$$A'_1B_1 + \ldots + A'_nB_n + C''B_{n+1} = B'_1B_1 + \ldots + B'_nB_n + C'B_{n+1}$$

We think of this subgoal as "balanced" because they have this common tail form, and all their heads have bounded size.

Introducing balanced subgoals is not sufficient for showing decidability. For the sizes of the common tails may keep growing. There is one more tableau rule, CUT, which allows one to cut the common tails. The exact formulation relies on families of auxiliary nonterminals ranged over by V, each of which has an associated definition $V \stackrel{\text{def}}{=} B$. We say that $(V_1, \ldots, V_n)$ is a family of recursive nonterminals if for each i either $V_i \stackrel{\text{def}}{=} A_{i1}V_1 + \ldots + A_{in}V_n$ where $A_{i1} + \ldots + A_{in}$ is admissible and does not contain auxiliary nonterminals, or $V_i \stackrel{\text{def}}{=} V_j$ and $j \leq i$ and $V_j \stackrel{\text{def}}{=} V_j$. An auxiliary nonterminal can only appear as a final element in a sequence of nonterminals. Admissibility is extended to such families of sequences. A configuration which is a singleton V is admissible and $\beta_1V'_1 + \ldots + \beta_kV'_k$ is admissible if the head $\beta_1 + \ldots + \beta_k$ is admissible and each β_i is distinct, and there is a family of recursive nonterminals $(V_1, \ldots, V_n)$ such that each V'_i is one of the V_js. An admissible configuration can therefore be presented in tail form

$A = A_1V_1 + \ldots + A_nV_n$. The definition of $(A \cdot w)$ is refined. If $(A_i \cdot w) = \epsilon$ and $V_i \stackrel{\text{def}}{=} B$ then $(A \cdot w) = B$. The language accepted by A is the set of words w such that $(A \cdot w) = V_i$ where $V_i \stackrel{\text{def}}{=} V_i$. Two configurations containing auxiliary nonterminals are equivalent if they accept the same words and agree on their terminating nonterminals.

The idea of CUT is that a balanced goal

$$(1) \quad A_1B_1 + \ldots + A_nB_n = C_1B_1 + \ldots + C_nB_n$$

where the A_is and C_is do not contain recursive nonterminals, can be reduced to a subgoal of the form

$$(2) \quad A_1V_1 + \ldots + A_nV_n = C_1V_1 + \ldots + C_nV_n$$

where $(V_1, \ldots, V_n)$ is a family of recursive nonterminals. The mechanism for reducing goal (1) to goal (2) involves constructing the recursive family $(V_1, \ldots, V_n)$ from a subsidary family of goals, $A_1^iB_1 + \ldots + A_n^iB_n = C_1^iB_1 + \ldots + C_n^iB_n$ where $i \geq 1$, with the same tails as (1).

We now state an important result which underpins the rule CUT.

Lemma 1 *Assume* $0 < m \leq n$. *If for all* $i : 1 \leq i \leq m$, $L(A_1^iB_1 + \ldots + A_n^iB_n) = L(C_1^iB_1 + \ldots + C_n^iB_n)$ *then there is a family of recursive nonterminals* $(V_1, \ldots, V_n)$ *such that*

1. *For each* $i : 1 \leq i \leq m$, $L(A_1^iV_1 + \ldots + A_n^iV_n) = L(C_1^iV_1 + \ldots + C_n^iV_n)$,
2. *If* $V_j \stackrel{\text{def}}{=} A_1'V_1 + \ldots + A_n'V_n$ *then* $L(B_j) = L(A_1'B_1 + \ldots + A_n'B_n)$,
3. *If* $V_i \stackrel{\text{def}}{=} V_j$ *then* $L(B_i) = L(B_j)$.

The recursive family $(V_1, \ldots, V_n)$ which issues from the proof of Lemma 1 is said to be "canonical" for the family $A_1^iB_1 + \ldots + A_n^iB_n = C_1^iB_1 + \ldots + C_n^iB_n$ of true goals. The construction of canonical nonterminals is independent of the tails B_i.

Fact 1 *If* $(V_1, \ldots, V_n)$ *is canonical for* $A_1^iB_1 + \ldots + A_n^iB_n = C_1^iB_1 + \ldots + C_n^iB_n$ *then it is also canonical for the family* $A_1^iD_1 + \ldots + A_n^iD_n = C_1^iD_1 + \ldots + C_n^iD_n$, *where* $i : 1 \leq i \leq k$.

The proof of Lemma 1 assembles the canonical family in stages. At stage j, the family $(V_1^{j+1}, \ldots, V_n^{j+1})$ is constructed from $(V_1^j, \ldots, V_n^j)$. If each V_i^{j+1} has the same definition as V_i^j then the construction terminates. In fact it must terminate by stage $j = n - 1$. The building of the V_i^{j+1}s from the V_i^js appeals to a smallest distinguishing word u_{j+1} for $L(A') \neq L(C')$, where A' is $A_1^lV_1^j + \ldots + A_n^lV_n^j$ and C' is $C_1^lV_1^j + \ldots + C_n^lV_n^j$ for some l. The depth of a canonical family is given by the sum over all stages of the distinguishing words, $\sum |u_j|$.

We need to consider how to introduce recursive nonterminals when the family of goals need not all be true. The idea is to approximate canonicity by defining when a recursive family $(V_1, \ldots, V_n)$ is "canonical to depth d" where $d \geq 0$, for a

family of goals $A_1^i B^1 + \ldots + A_n^i B_n = C_1^i B_1 + \ldots + C_n^i B_n$. The construction is the same as for the proof of Lemma 1, except that we stop at the first stage $j \geq 0$ with $(V_1^j, \ldots, V_n^j)$ as the required family of recursive nonterminals if the sum of the distinguishing words $s_j = |u_1| + \ldots + |u_j|$ is no larger than d, and for all w such that $|w| \leq d - s_j$, $w \in L(A_1^i V_1^j + \ldots + A_n^i V_n^j)$ iff is $w \in L(C_1^i V_1^j + \ldots + C_n^i V_n^j)$, for each i.

The rule CUT, where $k \leq n$, is as follows.

$$\frac{\begin{array}{r} A_1^1 B_1 + \ldots + A_n^1 B_n = C_1^1 B_1 + \ldots + C_n^1 B_n \\ \vdots \\ A_1^k B_1 + \ldots + A_n^k B_n = C_1^k B_1 + \ldots + C_n^k B_n \\ \vdots \\ A_1 B_1 + \ldots + A_n B_n = C_1 B_1 + \ldots + C_n B_n \end{array} \quad \mathcal{C}}{A_1 V_1 + \ldots + A_n V_n = C_1 V_1 + \ldots + C_n V_n}$$

where $\mathcal{C}$ is the condition that $(V_1, \ldots, V_n)$ is canonical to depth d for the family of goals $A_1^i B_1 + \ldots + A_n^i B_n = C_1^i B_1 + \ldots + C_n^i B_n$, $1 \leq i \leq k$, and there are at least d applications of UNF (as well as possibly applications of BAL) between $A_1^k B_1 + \ldots + A_n^k B_n = C_1^k B_1 + \ldots + C_n^k B_n$ and the final goal in the premises $A_1 B_1 + \ldots + A_n B_n = C_1 B_1 + \ldots + C_n B_n$. CUT is essentially the strategy T_C in S'enizuergues's proof (although he uses regular expressions and not recursive nonterminals).

From Fact 1 it follows that for any other family of goals with different tails D_i but the same heads A_j^i, C_j^i the same recursive nonterminal family is introduced. It is this feature which guarantees that there is a finite tableau proof for $F \sim G$.

We have now seen all the tableau proof rules, INIT, UNF, BAL(L), BAL(R) and CUT. There is also the important notion of when a current goal counts as final. Final goals are classified as either successful or unsuccessful. A *tableau proof* for the starting goal $F = G$ is a finite proof tree, whose root is $F = G$ and all of whose leaves are successful final goals, and all of whose inner subgoals are the result of an application of one of the rules. Successful final goals are as follows:

$$\begin{array}{ll} & A = B \\ & \vdots \qquad \text{UNF at least once} \\ A = A & A = B \end{array}$$

An identity and a goal which is repeated count as successful. Unsuccessful final goals are

$$\emptyset = B \text{ and } L(B) \neq \emptyset \qquad A = \emptyset \text{ and } L(A) \neq \emptyset \qquad V_i = V_j \text{ and } i \neq j$$

The tableau rules are sound and complete, which we now explain. First UNF is complete in the sense that if the premise is true then so are the subgoals. Completeness for BAL is that if the premise goals (those above the subgoal) are true then so is the subgoal. The statement of completeness for CUT is that there

are correct applications of it. If $(V_1, \ldots, V_n)$ is canonical for the first k premises then there is a depth d for which it is canonical. Moreover $(V_1, \ldots, V_n)$ needs to be a recursive family for the true goal $A_1B_1 + \ldots + A_nB_n = C_1B_1 + \ldots + B_nC_n$, in which case the subgoal follows.

For soundness of the tableau rules consider global soundness of the proof system. The overall idea is that if there is a successful tableau whose root is false then there is a path through the tableau within which each subgoal is false. The idea is refined using approximants. If $F^i \not\sim G^i$ then there is smallest distinguishing word w. One can define n-equivalence between F^i and G^i, if for all words w such that $|w| \leq n$, w does not distinguish between F^i and G^i. UNF obeys the simple soundness property that if the goal is not $n+1$-equivalent then a subgoal is not n-equivalent. Therefore if the root is false then there is an offending path (of false goals) through the tableau within which the approximant indices decrease whenever rule UNF has been applied, and hence this would mean that a successful final goal is false (which, as we shall show, is impossible). Soundness of the conditional rules is that if the premises are on an offending path then the subgoal preserves the falsity index of the goal immediately above it. In the case of BAL(R) assume that the offending path passes through the premise goals. There is a least n such that for the initial premise B is n-equivalent to $X_1A_1 + \ldots + X_kA_k$ and B is not $n+1$-equivalent to $X_1A_1 + \ldots + X_kA_k$. As there are exactly M applications of UNF between the initial and final premise it follows that B' is $(n - \mathrm{M})$-equivalent to $D_1A_1 + \ldots + D_kA_k$. However, as this is the offending path B' is not $(n + 1 - \mathrm{M})$-equivalent to $D_1A_1 + \ldots + D_kA_k$. A small argument shows that B' is not $(n + 1 - M)$-equivalent to $D_1(B \cdot w(X_1)) + \ldots + D_k(B \cdot w(X_k))$ (because A_i is $(n - \mathrm{M})$-equivalent to $(B \cdot w(X_i)$. There is a similar soundness argument for CUT. The idea of this style of soundness is essentially due to Sénizuergues (although he uses the different framework of deduction systems).

The main result is as follows, and a similar result holds for monadic resursion schemes.

Theorem 1 *$F \sim G$ iff there is a finite tableau proof for $F = G$.*

5 Conclusion

We have sketched decidability of equivalence for two old schema problems. However there are many open questions for further work. First we do not have a complexity bound for the decision procedures. Secondly we have only shown decidability for first-order recursion schemes. There is a known hierarchy of schema problems at higher order [3,4]. The branch languages of higher-order schemes are deterministic context-sensitive languages, as illustrated by the following 2nd-order scheme

$$\Phi(G, H)(x) \stackrel{\text{def}}{=} f(\Phi(Gg, Hh)(x), G(Hx))$$

starting from $\Phi(g, h)(x)$. And so little is known about deterministic context-sensitive languages.

References

1. Courcelle, B. (1978). A representation of trees by languages I, *Theoretical Computer Science*, **6**, 255-279.
2. Courcelle, B. (1978). A representation of trees by languages II, *Theoretical Computer Science*, **7**, 25-55.
3. Damm W, (1977). Languages defined by higher type program schemes, *Lecture Notes in Computer Science*, **52**, 164-179.
4. Damm W, (1979). An algebraic extension of the Chomsky-hierarchy. *Lecture Notes in Computer Science*, **74**, 266-276.
5. Friedman, E. (1977). Equivalence problems for deterministic context-free languages and monadic recursion schemes. *Journal of Computer and System Sciences*, **14**, 344-359.
6. Garland, S., and Luckham, D. (1973). Program schemes, recursion schemes, and formal languages. *Journal of Computer and System Sciences*, **7**, 119-160.
7. Harrison, M. (1978). *Introduction to Formal Language Theory*, Addison-Wesley.
8. Harrison, M., and Havel, I. (1973). Strict deterministic grammars. *Journal of Computer and System Sciences*, **7**, 237-277.
9. Hüttel, H., and Stirling, C. (1991). Actions speak louder than words: proving bisimilarity for context free processes. *Proceedings 6th Annual Symposium on Logic in Computer Science*,IEEE Computer Science Press, 376-386.
10. Sénizergues, G. (1997). The equivalence problem for deterministic pushdown automata is decidable. *Lecture Notes in Computer Science*, **1256**, 671-681.
11. Sénizergues, G. (1998). L(A) = L(B)? Tech. Report LaBRI, Université Bordeaux I, pp. 1-166. (To appear in *Theoretical Computer Science*.)
12. Stirling, C. (1998). Decidability of bisimulation equivalence for normed pushdown processes. *Theoretical Computer Science*, **195**, 113-131.
13. Stirling, C. (1999). Decidability of DPDA equivalence. Tech. Report LFCS-99-411, University of Edinburgh, pp. 1-25. (To appear in *Theoretical Computer Science*.)

Automated Verification = Graphs, Automata, and Logic

Moshe Y. Vardi*

Rice University, Department of Computer Science, Houston, TX 77005-1892, USA

Abstract. In automated verification one uses algorithmic techniques to establish the correctness of the design with respect to a given property. Automated verification is based on a small number of key algorithmic ideas, tying together graph theory, automata theory, and logic. In this self-contained talk I will describe how this "holy trinity" gave rise to automated-verification tools.

References

1. E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, January 1986.
2. E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.
3. R.P. Kurshan. *Computer Aided Verification of Coordinating Processes*. Princeton Univ. Press, 1994.
4. M.Y. Vardi. An automata-theoretic approach to linear temporal logic. In F. Moller and G. Birtwistle, editors, *Logics for Concurrency: Structure versus Automata*, volume 1043 of *Lecture Notes in Computer Science*, pages 238–266. Springer-Verlag, Berlin, 1996.
5. M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. First Symposium on Logic in Computer Science*, pages 332–344, Cambridge, June 1986.

* Supported in part by NSF grant CCR-9700061, and by a grant from the Intel Corporation. URL: `http://www.cs.rice.edu/~vardi`.

P. Clote and H. Schwichtenberg (Eds.): CSL 2000, LNCS 1862, p. 139, 2000.

A Fully Complete PER Model for ML Polymorphic Types

Samson Abramsky[1] and Marina Lenisa[2] *

[1] LFCS, Division of Informatics, University of Edinburgh,
The King's Buildings, Mayfield Road, Edinburgh EH9 3JZ, UK.
samson@dcs.ed.ac.uk
[2] Dipartimento di Matematica e Informatica, Università di Udine,
Viale delle Scienze 206, 33100 Udine, ITALY.
lenisa@dimi.uniud.it

Abstract. We present a *linear realizability* technique for building *Partial Equivalence Relations (PER)* categories over *Linear Combinatory Algebras*. These PER categories turn out to be *linear categories* and to form an *adjoint model* with their *co-Kleisli categories*. We show that a special linear combinatory algebra of *partial involutions*, arising from *Geometry of Interaction* constructions, gives rise to a *fully* and *faithfully complete* model for *ML polymorphic types* of system F.

Keywords: ML-polymorphic types, linear logic, PER models, Geometry of Interaction, full completeness.

Introduction

Recently, Game Semantics has been used to define fully-complete models for various fragments of Linear Logic ([AJ94a, AM99]), and to give fully-abstract models for many programming languages, including PCF [AJM96, HO96, Nic94], richer functional languages [McC96], and languages with non-functional features such as reference types and non-local control constructs [AM97, Lai97].

All these results are crucially based on the *linear* analysis of the *intuitionistic* arrow which is possible in the *intensional* setting of game categories. However, the definitions of game and game categories are quite complex, often requiring cumbersome quotienting operations. In this paper, we present the technique of *linear realizability* as a simpler and more direct alternative to game constructions for addressing *full completeness* issues.

The linear realizability technique amounts to constructing a category of *Partial Equivalence Relations (PERs)* over a *Linear Combinatory Algebra (LCA)*, which turns out to be a *linear category*, and to form an *adjoint model* with its co-Kleisli category. The notion of Linear Combinatory Algebra introduced by the first author ([Abr97a]) refines the standard notion of Combinatory Algebra, in the same way in which intuitionistic linear logic refines intuitionistic logic. The

* Work partially supported by TMR Linear FMRX-CT98-0170.

P. Clote and H. Schwichtenberg (Eds.): CSL 2000, LNCS 1862, pp. 140–155, 2000.

construction of PER models from LCA's presented in this paper is quite simple and clear, and it yields models with *extensionality* properties, thus avoiding the quotienting operations which are often needed in defining game categories and models. Moreover, PER categories offer simple natural models for second order (polymorphic) λ-calculus, i.e. Girard's System F ([Gir72]).

Recently, there has been much interest in realizability techniques, and in particular in *linear* realizability, especially in connection with full completeness and full abstraction problems. Realizability can be regarded as a powerful tool for mediating between *intensional* and *extensional* aspects of computation, and it has been used for extensionalizing intensional constructions (e.g. in [AM99]), and as a technique for building directly interesting (possibly fully-complete/fully-abstract) models. Examples of this latter use of realizability appear in this paper, and in [AL99], where a fully-abstract PER model for PCF, alternative to the game model of [AJM96], is provided using the algebra of *well-bracketed strategies*.

A *categorical* model of a type theory (or logic) is said to be *fully-complete* ([AJ94a]) if, for all *types* (*formulae*) A, B, all morphisms $f : [\![A]\!] \to [\![B]\!]$, from the interpretation of A into the interpretation of B, are denotations of a *proof-term* of the entailment $A \vdash B$, i.e. if the interpretation function from the category of syntactical objects to the category of denotations is *full*. The notion of full-completeness is the counterpart of the notion of *full abstraction*, in the sense that, if the term language is executable, then a fully-complete model is (up-to a possible quotient) fully-abstract.

Besides full completeness, one can ask the question whether the theory induced by a model $\mathcal{M}$ coincides precisely with the syntactical theory or whether more equations are satisfied in $\mathcal{M}$. A model $\mathcal{M}$ is called *faithful* if it realizes exactly the syntactical theory.

The fully and faithfully complete model for ML-types built in this paper is obtained as an instance of the PER construction, by considering the special linear combinatory algebra of *partial involutions*. ML-types are universal closures of simple types, i.e. types of the form $\forall X_1 \ldots X_n.T$, where T is $\forall$-free and $FV(T) \subseteq \{X_1, \ldots, X_n\}$. The algebra of partial involutions arises in the context of the generalization of Girard's *Geometry of Interaction* due to the first author ([AJ94,Abr96,Abr97a,AHPS98]). This is a powerful construction, which allows to build many new combinatory algebras, as well as to recover previously known models by viewing them in an alternative perspective. The algebra of partial involutions is a highly constrained algebra, in which all computations are *reversible*. Partial involutions are reminiscent of the *copy-cat* strategies of game categories, in that all the combinators mediate the required interactions between the arguments simply by copying information between input and output ports.

The proof of full completeness consists in showing that this model satisfies the axioms in the axiomatization of fully-complete models for ML-types given in [AL99a]. This axiomatization is given on the models of system F which are called *hyperdoctrines* ([Cro93]). In particular, it works in the context of *adjoint models*. It consists of two main steps. The first is an axiomatization of the fact that every morphism $f : 1 \to [\![T]\!]$, where T is an ML-type generates, under

decomposition, a possibly *infinite* typed Böhm tree. Then, an axiom which rules out infinite trees from the model is introduced.

Proving that the model of partial involutions considered in this paper does not contain infinite typed Böhm trees is quite difficult, and it requires the study of an intermediate model. This is the model generated by the *Sierpinski PER* and it consists of all (possibly infinite) Böhm trees of the typed λ-calculus, with constants $\bot, \top$. A crucial step in our proof consists in proving that, in the simply typed λ-calculus with *typical ambiguity* and $\bot$-constants, "totality tests" are λ-definable by finite typed trees. These totality tests allow us to tell apart normal forms in which $\bot$ appears, from those in which $\bot$ does not appear. A further ingredient is an *Approximation Lemma*, along the lines of [AJM96].

The full completeness result obtained in this paper is interesting, since, until now, the research on full completeness for System F has produced fully-complete *denotational* models only for a small subclass of ML-types, i.e. the *algebraic types* (see [HRR90]). In the literature, there are two fully-complete models for the whole system F: i.e. that of [BC88], and that of [Hug99]. The first model is based on a quotient of a term model, the latter is a game model. But both these models still have a somewhat syntactical flavor, and their constructions are extremely complex. The model in this paper can be viewed as the first *denotational* model which is fully-complete for the whole class of ML-types.

In Section 1, we recall the syntax of ML types of system F, and we present two results on the simply typed λ-calculus with a theory satisfying *Typical Ambiguity*. The first is due to Statman, the latter is a new *Typed Separability* result. In Section 2, we recall the notion of $2\lambda\times$-hyperdoctrine and the notion of *adjoint hyperdoctrine* introduced in [AL99a], and we formalize the definition of fully-complete hyperdoctrine. In Section 3, we present the linear realizability technique, for building PER categories over LCAs. In Section 4, the LCA of partial involutions is described. In Section 5, the proof of full completeness for the PER model over the LCA of partial involutions is sketched. Final remarks and directions for future work appear in Section 6.

The authors are thankful to F.Honsell, R.Jagadeesan, J.Laird, J.Longley, S.Martini, G.Plotkin, A.Simpson for useful discussions on some of the issues of the paper.

1 ML Polymorphism

First, we recall the syntax of the class of *ML-types* of system F. Then, we present two important results on the simply typed λ-calculus with a theory satisfying *Typical Ambiguity*. A theory is said to satisfy Typical Ambiguity if two terms are equated if and only if they are equated for all possible substitutions of type variables. The first result that we present is Statman's *Typical Ambiguity Theorem*, which asserts that there is exactly one consistent theory satisfying Typical Ambiguity on the simply typed λ-calculus with infinitely many type variables: this is the $\beta\eta$-theory. An immediate consequence of this result is that the only consistent theory on the fragment of system F consisting of *ML-types* is precisely

the $\beta\eta$-theory. The second result concerns the definability of "convergence tests" in the simply typed λ-calculus with infinitely many type variables, $\bot$-constants, and satisfying Typical Ambiguity. In particular, we prove that, for any given type, there are convergence test terms, which detect the presence of $\bot$-constants in a term of that type. This implies immediately that, in a theory of Typical Ambiguity over the simply typed λ-calculus with infinite type variables and $\bot$-constants, a term containing $\bot$ in its normal form can never be equated to a term in whose normal form $\bot$ does not appear. This result is used in the proof of full completeness of the model of PERs over the LCA of partial involutions.

We assume that the reader is familiar with System F (see e.g. [AL91]).

The class of *ML-polymorphic types* of system F corresponds to the limited kind of polymorphism allowed in the language ML.

Definition 1 (ML-types). *The class* ML-Type *of ML-types is defined by:*

$$\textit{ML-Type} = \{\forall \boldsymbol{X}.T \mid T \in \ \textit{SimType} \ \wedge \ FV(T) \subseteq \boldsymbol{X}\} \ ,$$

where SimType *is the class of* simple types *of system F, i.e. simple types over an infinite set of type variables, and* $\boldsymbol{X}$ *stands for* $X_1, \ldots, X_n$*, for* $n \geq 0$.

Terms of ML-types have essentially the same "combinatorics" as the typically ambiguous terms of the simply typed λ-calculus. In fact, any theory on ML-terms induces a theory satisfying Typical Ambiguity.

The following is a result about simply typed λ-calculus with *infinitely many* type variables λ^∞, first proved in [Sta88].

Theorem 1 (Statman's Typical Ambiguity). *Let* T *be a type of* λ^∞ *s.t.* $FV(T) \subseteq \{X_1, \ldots, X_n\}$. *If* $\forall\ M =_{\beta\eta} N \ : \ T$*, then, there exist types* $S_1, \ldots, S_n$*, and* $Y \in TVar$*, and a term* L *s.t.* $\vdash L[\boldsymbol{S}/\boldsymbol{X}] : T[\boldsymbol{S}/\boldsymbol{X}] \to Bool_Y$*, where* $Bool_Y = Y \to Y \to Y$*, s.t.*

$\vdash (LM)[\boldsymbol{S}/\boldsymbol{X}] =_{\beta\eta} true \ : \ Bool_Y \ \wedge \ \vdash (LN)[\boldsymbol{S}/\boldsymbol{X}] =_{\beta\eta} false \ : \ Bool_Y$,
where $true = \lambda x : Y.y : Y.x$ *and* $false = \lambda x : Y.y : Y.y$.

Corollary 1. *i) The maximal consistent theory satisfying Typical Ambiguity on the simply typed* λ*-calculus with infinitely many type variables is the* $\beta\eta$*-theory. ii) The maximal consistent theory on the fragment of system F consisting of ML-types is the* $\beta\eta$*-theory.*

As it will be clear in the following section from the definition of full completeness, by Corollary 1ii), any non-trivial fully-complete model for ML-types of system F is necessarily *faithful*, i.e. it realizes exactly the $\beta\eta$-theory at ML-types.

Now we show that "convergence tests" are λ-definable in the simply typed λ-calculus with infinitely many type variables and $\bot$-constants for any type variable, and a theory satisfying Typical Ambiguity, which we call $\lambda^\infty_\bot$.

Definition 2 (Typed Convergence Tests). *Let* $T = T_1 \to \ldots \to T_n \to X_k \in SimType$*, let* ι *be a distinguished type variable, and let* $\alpha_T = T[\iota \to \iota/\boldsymbol{X}]$. *We define, by induction on* T*, the* convergence test *term* $\vdash S_{\alpha_T} : \alpha_T$ *as follows:*
if $T = X$*, then* $S_{\iota\to\iota} = I_{\iota\to\iota}$, *otherwise,*
let $T = T_1 \to \ldots \to T_n \to X_k$*, where* $T_i = U_{i1} \to \ldots \to U_{iq_i} \to X_i$*, then*
$S_{\alpha_T} = \lambda x_1 : \alpha_{T_1} \ldots x_n : \alpha_{T_n}.\lambda z : \iota.(x_1 S_{\alpha_{U_{11}}} \ldots S_{\alpha_{U_{1q_1}}})(\ldots (x_n S_{\alpha_{U_{n1}}} \ldots S_{\alpha_{U_{nq_n}}} z)) \ .$

The "convergence test" terms defined above give us a procedure for deciding whether a normal form of $\lambda_\perp^\infty$ contains a divergent subterm. Namely, let M be a normal form of $\lambda_\perp^\infty$ of type $T_1 \to \ldots \to T_n \to X_k$. We first instantiate all the free variables in M by $\iota \to \iota$, then we apply M to the sequence of convergence tests $S_{\alpha_{T_1}}, \ldots, S_{\alpha_{T_n}}$. The effect of this is that, in the head reduction of $MS_{\alpha_{T_1}}, \ldots, S_{\alpha_{T_n}}$, each subterm of M definitely appears in head position, and it reduces to the identity, until a $\perp$ is detected.

For a term $\boldsymbol{y} : \boldsymbol{U} \vdash M : T$, we denote by $\boldsymbol{y} : \boldsymbol{\alpha_U} \vdash M_{\alpha_T} : \alpha_T$ (or simply by M_{α_T}) the term of type α_T obtained from $\boldsymbol{y} : \boldsymbol{U} \vdash M : T$ by instantiating all the type variables free in T by $\iota \to \iota$.

Theorem 2 (Typed Separability). *Let $T = T_1 \to \ldots \to T_n \to X_k \in SimType$, and let $\vdash M : T$ be a term of $\lambda_\perp^\infty$. Then*

$$M_{\alpha_T} S_{\alpha_{T_1}} \ldots S_{\alpha_{T_n}} = \begin{cases} I_{\iota\to\iota} & \text{if the normal form of } M \text{ is } \perp\text{-free} \\ \lambda x : \iota.\ \perp & \text{otherwise .} \end{cases}$$

Theorem 2 above can be regarded as a *typed* Böhm-like Separability Theorem, in the sense that, if we think of $\perp$ as a generic *unsolvable* term, then Theorem 2 allows us to tell apart normal forms from unsolvable terms.

Corollary 2. *In any theory satisfying Typical Ambiguity on $\lambda_\perp^\infty$, a term in whose normal form $\perp$ appears cannot be equated to a term, in whose normal form $\perp$ does not appear.*

2 Models of System F

We recall first the notion of $2\lambda\times$-hyperdoctrine (see [Cro93]). This essentially corresponds to the notion of *external model* (see [AL91]). Then, we give the formal definition of fully (and faithfully) complete hyperdoctrine model. Finally, we define the categorical notion of *adjoint hyperdoctrine*, on which the axiomatization of full completeness at ML-types of [AL99a] is given. Adjoint hyperdoctrines arise as *co-Kleisli* indexed categories of *linear* indexed categories.

In what follows, we assume that all indexed categories which we consider are strict (see e.g. [AL91,Cro93] for more details).

Definition 3 ($2\lambda\times$-hyperdoctrine). *A* $2\lambda\times$-hyperdoctrine *is a triple* $(\mathcal{C}, \boldsymbol{G}, \forall)$, *where:*

- $\mathcal{C}$ *is the* base category, *it has with finite products, and it consists of a distinguished object $\mathcal{U}$ which generates all other objects using the product operation $\times$. We will denote by $\mathcal{U}^m$, for $m \geq 0$, the objects of $\mathcal{C}$.*
- $\boldsymbol{G} : \mathcal{C}^{op} \to$ CCCat *is a $\mathcal{C}$-indexed cartesian closed category, where* CCCat *is the category of cartesian closed categories and strict cartesian closed functors, such that: for all $\mathcal{U}^m$, the collection of objects of the cartesian closed fibre category $\boldsymbol{G}(\mathcal{U}^m)$ is indexed by the morphisms from $\mathcal{U}^m$ to $\mathcal{U}$ in $\mathcal{C}$, i.e. the objects of $\boldsymbol{G}(\mathcal{U}^m)$ are the morphisms in $Hom_{\mathcal{C}}(\mathcal{U}^m, \mathcal{U})$, and, for any $f : \mathcal{U}^m \to \mathcal{U}^n$ in $\mathcal{C}^{op}$, the cartesian closed functor $\boldsymbol{G}(f) : \boldsymbol{G}(\mathcal{U}^n) \to \boldsymbol{G}(\mathcal{U}^m)$, called* reindexing functor *and denoted by f^*, is s.t., for any object $h : \mathcal{U}^n \to \mathcal{U}$, $f^*(h) = f; h$;*

– *For each object* $\mathcal{U}^m$ *of* $\mathcal{C}$*, there are functors* $\forall_m : \boldsymbol{G}(\mathcal{U}^m \times \mathcal{U}) \to \boldsymbol{G}(\mathcal{U}^m)$ *s.t.*
  - $\forall_m$ *is right adjoint to the functor* $\pi_m^* : \boldsymbol{G}(\mathcal{U}^m) \to \boldsymbol{G}(\mathcal{U}^m \times \mathcal{U})$*, where* $\pi_m : \mathcal{U}^m \times \mathcal{U} \to \mathcal{U}^m$ *is the projection in* $\mathcal{C}$*;*
  - $\forall_m$ *satisfies the* Beck-Chevalley condition.

Any $2\lambda\times$-hyperdoctrine can be endowed with a notion of interpretation $[\![\ ]\!]$ for the language of system F.
Types with free variables in $X_1, \ldots, X_m$ are interpreted by objects of $\mathbf{G}(\mathcal{U}^m)$, i.e. by morphisms from $\mathcal{U}^m$ to $\mathcal{U}$ in $\mathcal{C}$: $[\![X_1, \ldots, X_m \vdash T]\!] : \mathcal{U}^m \to \mathcal{U}$.
Well-typed terms, i.e. $X_1, \ldots, X_m; x_1 : T_1, \ldots, x_n : T_n \vdash M : T$, are interpreted by morphisms in the category $\mathbf{G}(\mathcal{U}^m)$:
$[\![X_1, \ldots, X_m; x_1 : T_1, \ldots, x_n : T_n \vdash M : T]\!] : [\![\boldsymbol{X} \vdash T_1]\!] \times \ldots \times [\![\boldsymbol{X} \vdash T_n]\!] \to [\![\boldsymbol{X} \vdash T]\!]$.

Definition 4 (Full (and Faithful) Completeness). *Let* $\mathcal{M} = (\mathcal{C}, \boldsymbol{G}, \forall, [\![\ ]\!])$ *be a* $2\lambda\times$*-hyperdoctrine.* $\mathcal{M}$ *is* fully and faithfully complete *w.r.t. the class of closed types* $\mathcal{T}$ *if, for all* $T \in \mathcal{T}$*,*
$\forall f \in Hom_{\boldsymbol{G}(1)}(1, [\![\vdash T]\!])$. $\exists(!)\beta\eta$*-normal form* M. $\vdash M : T \ \wedge \ f = [\![\vdash M : T]\!]$.

In the following definition, we capture those $2\lambda\times$-hyperdoctrines which arise from a co-Kleisli construction over an *indexed linear category*, and on which the axiomatization of fully-complete models for ML-types given in [AL99a] is based.

Definition 5 (Adjoint Hyperdoctrine).
An adjoint hyperdoctrine *is a quadruple* $(\mathcal{C}, \boldsymbol{L}, \boldsymbol{G}, \forall)$*, where:*
– $\mathcal{C}$ *is the* base category*, it has finite products, which consists of a distinguished object* $\mathcal{U}$ *which generates all other objects using the product operation* $\times$*. We will denote by* $\mathcal{U}^m$*, for* $m \geq 0$*, the objects of* $\mathcal{C}$*.*
– $\boldsymbol{L} : \mathcal{C}^{op} \to$ LCat *is a* $\mathcal{C}$*-indexed linear category, where* LCat *is the category of linear categories and strict monoidal closed functors preserving the comonad structure, s.t.: for all* $\mathcal{U}^m$*, the underlying collection of objects of the linear fibre category* $\boldsymbol{L}(\mathcal{U}^m)$ *is indexed by the morphisms from* $\mathcal{U}^m$ *to* $\mathcal{U}$ *in* $\mathcal{C}$*.*
– $\boldsymbol{G} : \mathcal{C}^{op} \to$ CCCat *is the* $\mathcal{C}$*-indexed co-Kleisli category of* $\boldsymbol{L}$*, which we assume to be cartesian closed.*
– *For each object* $\mathcal{U}^m$ *of* $\mathcal{C}$*, there are functors* $\forall_m : \boldsymbol{G}(\mathcal{U}^m \times \mathcal{U}) \to \boldsymbol{G}(\mathcal{U}^m)$ *s.t.*
  - $\forall_m : \boldsymbol{G}(\mathcal{U}^m \times \mathcal{U}) \to \boldsymbol{G}(\mathcal{U}^m)$ *is right adjoint to the functor* $\boldsymbol{G}(\pi_m) : \boldsymbol{G}(\mathcal{U}^m) \to \boldsymbol{G}(\mathcal{U}^m \times \mathcal{U})$*, where* $\pi_m : \mathcal{U}^m \times \mathcal{U} \to \mathcal{U}^m$ *is the projection in* $\mathcal{C}$*;*
  - $\forall_m : \boldsymbol{G}(\mathcal{U}^m \times \mathcal{U}) \to \boldsymbol{G}(\mathcal{U}^m)$ *satisfies the* Beck-Chevalley condition.

3 Models of PERs over a Linear Combinatory Algebra

Canonical examples of $2\lambda\times$-hyperdoctrines arise from considering the *Partial Equivalence Relation* (PER) category over a *combinatory algebra* (see [Cro93], Chapter 5, Section 5.5 for more details). In this section, we show how to build a PER category from a linear combinatory algebra (LCA). This category turns out to form an adjoint model with its co-kleisli category, and it gives rise to an adjoint hyperdoctrine.
We start by recalling the definition of linear combinatory algebra ([Abr97a], [AHPS98]):

Definition 6 (Linear Combinatory Algebra). *A* linear combinatory algebra $\mathcal{A} = (A, \bullet, !)$ *is an applicative structure* $(A, \bullet)$ *with a unary (injective) operation* $!$*, and distinguished elements (combinators)* $B, C, I, K, W, D, \delta, F$ *satisfying the following equations:*

Equation	**Principal type**	**Logical rule**
$\boldsymbol{I}x = x$	$\alpha \multimap \alpha$	Identity
$\boldsymbol{B}xyz = x(yz)$	$(\alpha \multimap \beta) \multimap (\gamma \multimap \alpha) \multimap \gamma \multimap \beta$	Cut
$\boldsymbol{C}xyz = (xz)y$	$(\alpha \multimap \beta \multimap \gamma) \multimap \beta \multimap \alpha \multimap \gamma$	Exchange
$\boldsymbol{K}x!y = x$	$\alpha \multimap !\beta \multimap \alpha$	Weakening
$\boldsymbol{W}x!y = x!y!y$	$(!\alpha \multimap !\alpha \multimap \beta) \multimap !\alpha \multimap \beta$	Contraction
$\boldsymbol{D}!x = x$	$!\alpha \multimap \alpha$	Dereliction
$\delta!x = !!x$	$!\alpha \multimap !!\alpha$	Comultiplication
$\boldsymbol{F}!x!y = !(xy)$	$!(\alpha \multimap \beta) \multimap !\alpha \multimap !\beta$	Closed Functoriality .

LCA's correspond to *Hilbert style* axiomatization of $\multimap, !$ fragment of Linear Logic. Given an LCA $\mathcal{A} = (A, \bullet, !)$, we can form a standard CA $\mathcal{A}_s = (A, \bullet_s)$ by the "combinatory version" of Girard's translation of Intuitionistic Logic into Linear Logic. We define: $\alpha \bullet_s \beta = \alpha \bullet !\beta$ (standard combinators can be defined in terms of the linear ones, see [AHPS98] for details).

We recall that a *BCI-algebra* is an applicative structure $(A, \bullet)$ with B, C, I combinators. In the next definition, we define a PER category over a BCI-algebra, which turns out to be symmetric monoidal closed.

Definition 7. *Let* $\mathcal{A} = (A, \bullet)$ *be a BCI-algebra. We define the category* $PER_{\mathcal{A}}$ *as follows.*
Objects*: PERs* $\mathcal{R} \subseteq A \times A$*, i.e. symmetric and transitive relations.*
Morphisms*: a morphism* f *from* $\mathcal{R}$ *to* $\mathcal{S}$ *is an equivalence class of the PER* $\mathcal{R} \multimap \mathcal{S}$*, where the PER* $\mathcal{R} \multimap \mathcal{S}$ *is defined by*

$$\alpha(\mathcal{R} \multimap \mathcal{S})\beta \quad \textit{iff} \quad \forall \gamma\, \mathcal{R}\, \gamma'.\ \alpha \bullet \gamma\, \mathcal{S}\, \beta \bullet \gamma' \ .$$

On BCI-algebras, standard *pairing* gives rise to a *tensor product*, but the definition of tensor product requires some care:

Lemma 1. *Let* $\mathcal{A} = (A, \bullet)$ *be a BCI-algebra. Let* P *be the* pairing combinator*, i.e. (using* λ*-notation)* $P = \lambda xyz.zxy$*. Then, for all PERs* $\mathcal{R}, \mathcal{S}$*, let* $\mathcal{R} \otimes \mathcal{S}$ *be the PER defined as the* transitive closure *of the following relation:*

$$\mathcal{R} \otimes' \mathcal{S} = \{(P\alpha\beta, P\alpha'\beta') \mid \alpha\, \mathcal{R}\, \alpha' \ \wedge \ \beta\, \mathcal{S}\, \beta'\} \ .$$

Notice in particular that, if the BCI-algebra is *affine*, i.e. it is a BCK-algebra, then the relation $\mathcal{R} \otimes' \mathcal{S}$ is already transitive, since, using projections, we get: $P\alpha\beta = P\alpha'\beta' \implies \alpha = \alpha' \ \wedge \ \beta = \beta'$.

Proposition 1. *Let* $\mathcal{A} = (A, \bullet)$ *be a BCI-algebra. Then* $PER_{\mathcal{A}}$ *is a symmetric monoidal closed category.*

Now we show how an LCA gives rise to a linear category.

Proposition 2. *Let $\mathcal{A} = (A, \bullet, !)$ be an LCA. Let $\boldsymbol{!} : PER_{\mathcal{A}} \to PER_{\mathcal{A}}$ be the functor defined by*

$$\forall\, \mathcal{R}\ .\ \boldsymbol{!}\,\mathcal{R} = \{(!\alpha, !\beta) \mid \alpha\ \mathcal{R}\ \beta\}, \quad \forall f : \mathcal{R}_1 \to \mathcal{R}_2\ .\ \boldsymbol{!}f = [F!f]\ .$$

Then $(\boldsymbol{!}, D, \delta, \phi, \phi')$ is a symmetric monoidal comonad, where

- $\phi_{\mathcal{R}_1,\mathcal{R}_2} : \boldsymbol{!}\,\mathcal{R}_1 \otimes \boldsymbol{!}\,\mathcal{R}_2 \to \boldsymbol{!}(\mathcal{R}_1 \otimes \mathcal{R}_2)$ *is defined by* $\phi_{\mathcal{R}_1,\mathcal{R}_2} = [\lambda u.F!P(uF)]$;
- $\phi' : I \simeq \boldsymbol{!}I$ *is* $[\delta]_{I \to \boldsymbol{!}I}$.

The following isomorphisms hold immediately in PER categories over LCA's:

Lemma 2. *Let $\mathcal{A} = (A, \bullet, !)$ be an LCA. Then, for all PERs $\mathcal{R}, \mathcal{S}$,*

1. (Idempotency of !) $[D] : \boldsymbol{!!}\,\mathcal{R} \simeq \boldsymbol{!}\,\mathcal{R} : [\delta]$;
2. (Uniformity of Threads) $\psi : \boldsymbol{!}\ \mathcal{R} \multimap \boldsymbol{!}\ \mathcal{S} \simeq \boldsymbol{!}\ \mathcal{R} \multimap \mathcal{S} : (\cdot)^\dagger$, *where* $\psi = [\lambda x.x; D]$; *or equivalently:* $\forall \alpha \in \boldsymbol{!}\,\mathcal{R} \multimap \boldsymbol{!}\,\mathcal{S}$, $(\alpha; [D])^\dagger = \alpha$;
3. (Commutativity of $\bigcap$ w.r.t. !) $\bigcap_X \boldsymbol{!}\,\mathcal{R} \simeq \boldsymbol{!}(\bigcap_X \mathcal{R})$.

The second isomorphism in Lemma 2 above is relevant for full completeness. In fact, this isomorphism amounts exactly to the *Uniformity of Threads Axiom* in the axiomatization of full completeness of [AL99a]. The isomorphisms of Lemma 2 above highlight the fact that the PER category is a "degenerate" model of linear logic.

Theorem 3. *Let $\mathcal{A} = (A, \bullet, !)$ be an LCA. Then*

- *The category $PER_{\mathcal{A}}$ is linear.*
- *The co-Kleisli category $(PER_{\mathcal{A}})_!$, induced by the comonad $\boldsymbol{!}$ on the category $PER_{\mathcal{A}}$, is cartesian closed.*
- *The categories $PER_{\mathcal{A}}$ and $(PER_{\mathcal{A}})_!$ form an adjoint model.*
- *The category $(PER_{\mathcal{A}})_!$ is isomorphic to the category $PER_{\mathcal{A}_s}$, where $PER_{\mathcal{A}_s}$ is the category obtained by standard realizability from the standard combinatory algebra $\mathcal{A}_s$.*

Finally, we show how to build an adjoint hyperdoctrine from an LCA:

Theorem 4 (PER Adjoint Hyperdoctrine). *Let $\mathcal{A} = (A, \bullet, !)$ be an LCA. Then $\mathcal{A}$ gives rise to an adjoint hyperdoctrine $(\mathcal{C}, \boldsymbol{L}, \boldsymbol{G}, \forall)$, by defining:*

$\mathcal{C}$: *Let $\mathcal{U}$ be the set $\{\mathcal{R} \mid \mathcal{R}$ is a PER on $A\}$. The objects of $\mathcal{C}$, $\mathcal{U}^n$, for $n \geq 0$, are the finite products in* Set *of n copies of the set $\mathcal{U}$, in particular $\mathcal{U}^0$ is the terminal object in* Set. *A morphism in $\mathcal{C}$, $f : \mathcal{U}^n \to \mathcal{U}^m$, is a set-theoretic function from $\mathcal{U}^m$ to $\mathcal{U}^n$.*

L : *The morphisms in the fibre category $\boldsymbol{L}(\mathcal{U}^m)$ from $h_1 : \mathcal{U}^m \to \mathcal{U}$ to $h_2 : \mathcal{U}^m \to \mathcal{U}$ are the equivalence classes of the PER $\bigcap_{\boldsymbol{X} \in \mathcal{U}^m} (h_1\boldsymbol{X} \multimap h_2\boldsymbol{X})$. For any object $f : \mathcal{U}^m \to \mathcal{U}$ in $\boldsymbol{L}(\mathcal{U}^m)$, we define $!f$ to be $\lambda \boldsymbol{X}.!(f\boldsymbol{X})$. For any morphism $f : \mathcal{U}^m \to \mathcal{U}^n$ in $\mathcal{C}$, we define the behavior of the functor $\boldsymbol{L}(f) : \boldsymbol{L}(\mathcal{U}^n) \to \boldsymbol{L}(\mathcal{U}^m)$ on morphisms by: for any morphism $H : h_1 \to h_2$ in $\boldsymbol{L}(\mathcal{U}^n)$, $H = \Lambda \boldsymbol{X}.H' \in \bigcap_{\boldsymbol{X}} (h_1\boldsymbol{X} \multimap h_2\boldsymbol{X})$, let $\boldsymbol{L}(f)(H) : \boldsymbol{L}(f)(h_1) \to \boldsymbol{L}(f)(h_2)$ be $\Lambda \boldsymbol{X}.H' \circ f(\boldsymbol{X}) \in \bigcap_{\boldsymbol{X}} (\boldsymbol{L}(f)(h_1)\boldsymbol{X} \multimap \boldsymbol{L}(f)(h_2)\boldsymbol{X})$.*

$\forall$: *The functor $\forall_m : \boldsymbol{L}(\mathcal{U}^m \times \mathcal{U}) \to \boldsymbol{L}(\mathcal{U}^m)$ is defined as follows. For any $h : \mathcal{U}^m \times \mathcal{U} \to \mathcal{U}$, $\forall_m(h) = \lambda \boldsymbol{X}.\bigcap_{\boldsymbol{X}} h(\boldsymbol{X})$. For any morphism $H : h_1 \to h_2$ in $\boldsymbol{L}(\mathcal{U}^m \times \mathcal{U})$, $\forall_m(H) = H$.*

4 Partial Involutions Affine Combinatory Algebra

Many examples of LCAs arise from the categorical version of Girard's Geometry of Interaction (GoI) construction, based on *traced symmetric monoidal* categories ([Abr97a,Abr96,AHPS98]). A basic example of GoI LCA, introduced in [Abr97a], can be defined on the space $[\mathbf{N} \rightharpoonup \mathbf{N}]$ of partial functions from natural numbers into natural numbers, by applying the GoI construction to the the traced category *Pfn* of sets and partial functions. Here we briefly recall the definition of this LCA, without discussing the categorical framework (see [Abr97a,Abr96,AHPS98] for more details). The LCA of partial involutions, which will be shown to provide a fully-complete model for ML-types (see Section 5), arises as subalgebra of this.

Let us consider the space $[\mathbf{N} \rightharpoonup \mathbf{N}]$ of partial functions from natural numbers to natural numbers. For any $\alpha \in [\mathbf{N} \rightharpoonup \mathbf{N}]$ injective, we denote by α^{-1} the inverse of α. Now we show how we can endow the space $[\mathbf{N} \rightharpoonup \mathbf{N}]$ with a structure of LCA. Actually, the algebra which we obtain is *affine*, i.e. it has a full $\boldsymbol{K}$-combinator. We start by fixing two injective *coding* functions t and p:

$$t : \mathbf{N} + \mathbf{N} \to \mathbf{N} \quad , \quad p : \mathbf{N} \times \mathbf{N} \to \mathbf{N} \ .$$

The first is used in order to define application, and it allows to transform an one-input/one-output function into a two-input/two-output function. The latter is used for creating *infinitely* many copies of an one-input/one-output function α, i.e. for defining $!\alpha$.

We now explain how application is computed *geometrically*, using the language of "boxes and wires" which arises in the general setting of traced symmetric monoidal categories (see [JSV96] for an abstract treatment).

Let us represent an one-input/one-output function $\alpha \in [\mathbf{N} \rightharpoonup \mathbf{N}]$ by the following one-input-port/one-output-port box (see Fig. 1(i) below).

In order to define the application $\alpha \bullet \beta$, for $\alpha, \beta \in [\mathbf{N} \rightharpoonup \mathbf{N}]$, we regard α as a two-input/two-output function via the coding t. In particular, $t; \alpha; t^{-1} : \mathbf{N}+\mathbf{N} \rightharpoonup \mathbf{N}+\mathbf{N}$ can be described as a matrix of 4 one-input/one-output functions, where each entry $\alpha_{ij} : \mathbf{N} \rightharpoonup \mathbf{N}$, $\alpha_{ij} = in_i; t; \alpha; t^{-1}; in_j^{-1}$ accounts for the contribution from the i-th input wire into the j-th output wire (see Fig. 1(ii)).

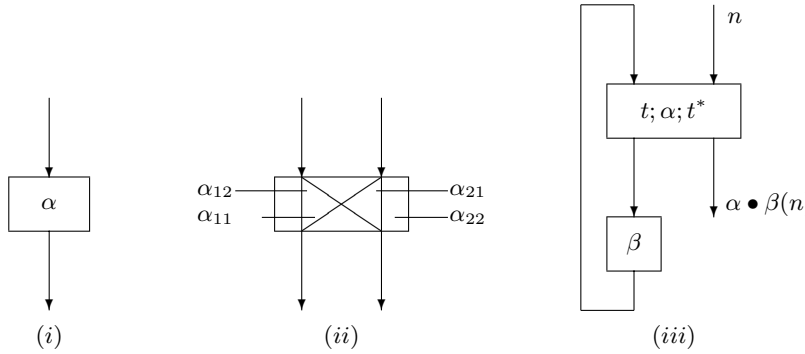


Fig. 1. Geometrical description of linear application.

The result of the application $\alpha \bullet \beta$ is the following one-input/one-output function (see Fig. 1(iii)):

$$\alpha \bullet \beta = \alpha_{22} \cup \alpha_{21}; (\beta; \alpha_{11})^\star; \beta; \alpha_{12} \ ,$$

where $\cup$ denotes union of graph relations, and $(\beta; \alpha_{11})^\star$ denotes $\bigcup_{n \geq 0} (\beta; \alpha_{11})^n$.

The above formula for computing the application is essentially the *Execution Formula* from Girard's Geometry of Interaction ([Gir89]).

The definition of the !-operation on our applicative structure is quite simple. The operation ! is intended to produce, from a single copy of α, *infinitely* many copies of α. These are obtained by simply tagging each of these copies with a natural number, i.e. we define:

$$!\alpha = p^{-1}; (id_{\mathbf{N}} \times \alpha); p \ .$$

Finally, we are left to show that (affine) combinators can be defined on the structure $([\mathbf{N} \rightharpoonup \mathbf{N}], \bullet, !)$. The formal (algebraic) definition of the combinators is the following:

Definition 8 (Combinators). *For* $\boldsymbol{X} \in \{\boldsymbol{I}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{K}, \boldsymbol{W}, \boldsymbol{D}, \delta, \boldsymbol{F}\}$*, let*

$$\boldsymbol{X} = s_{\boldsymbol{X}}^{-1}; f_{\boldsymbol{X}}; s_{\boldsymbol{X}} \ ,$$

where:

I :
- $s_{\boldsymbol{I}} = t$.
- $f_{\boldsymbol{I}} : \boldsymbol{N} + \boldsymbol{N} \rightharpoonup \boldsymbol{N} + \boldsymbol{N}$ *is defined by:* $\forall n.\ f_{\boldsymbol{I}}(r, n) = (l, n) \quad \wedge \quad \forall n.\ f_{\boldsymbol{I}}(l, n) = (r, n)$.

B :
- $s_{\boldsymbol{B}} : (((\boldsymbol{N} + \boldsymbol{N}) + (\boldsymbol{N} + \boldsymbol{N})) + \boldsymbol{N}) + \boldsymbol{N} \rightharpoonup \boldsymbol{N}$ *is defined by* $s_{\boldsymbol{B}} = ((t + t) + id_{\boldsymbol{N}}) + id_{\boldsymbol{N}}; (t + id_{\boldsymbol{N}}) + id_{\boldsymbol{N}}; t + id_{\boldsymbol{N}}; t$.
- $f_{\boldsymbol{B}} : (((\boldsymbol{N}+\boldsymbol{N})+(\boldsymbol{N}+\boldsymbol{N}))+\boldsymbol{N})+\boldsymbol{N} \rightharpoonup (((\boldsymbol{N}+\boldsymbol{N})+(\boldsymbol{N}+\boldsymbol{N}))+\boldsymbol{N})+\boldsymbol{N}$ *is the function defined by the following equations together with their symmetric closure:*
  - $\forall n.\ f_{\boldsymbol{B}}(r, n) = (l, (l, (l, (r, n))))$
  - $\forall n.\ f_{\boldsymbol{B}}(l, (l, (l, (l, n)))) = (l, (l, (r, (r, n))))$
  - $\forall n.\ f_{\boldsymbol{B}}(l, (l, (r, (l, n)))) = (l, (r, n))$.

C :
- $s_{\boldsymbol{C}} : ((\boldsymbol{N} + \boldsymbol{N}) + ((\boldsymbol{N} + \boldsymbol{N}) + \boldsymbol{N})) + \boldsymbol{N} \rightharpoonup \boldsymbol{N}$ *is defined by* $s_{\boldsymbol{C}} = (t + (t + id_{\boldsymbol{N}})) + id_{\boldsymbol{N}}; (t + t) + id_{\boldsymbol{N}}; t + id_{\boldsymbol{N}}; t$.
- $f_{\boldsymbol{C}} : ((\boldsymbol{N}+\boldsymbol{N})+((\boldsymbol{N}+\boldsymbol{N})+\boldsymbol{N}))+\boldsymbol{N} \rightharpoonup ((\boldsymbol{N}+\boldsymbol{N})+((\boldsymbol{N}+\boldsymbol{N})+\boldsymbol{N}))+\boldsymbol{N}$ *is the function defined by the following equations together with their symmetric closure:*
  - $\forall n.\ f_{\boldsymbol{C}}(r, n) = (l, (r, (r, n)))$
  - $\forall n.\ f_{\boldsymbol{C}}(l, (r, (l, (r, n)))) = (l, (l, (r, n)))$
  - $\forall n.\ f_{\boldsymbol{C}}(l, (r, (l, (l, n)))) = (l, (l, (l, n)))$.

K :
- $s_{\boldsymbol{K}} : (\boldsymbol{N} + \boldsymbol{N}) + \boldsymbol{N} \rightharpoonup \boldsymbol{N}$ *is defined by* $s_{\boldsymbol{K}} = t + id_{\boldsymbol{N}}; t$.
- $f_{\boldsymbol{K}} : (\boldsymbol{N} + \boldsymbol{N}) + \boldsymbol{N} \rightharpoonup (\boldsymbol{N} + \boldsymbol{N}) + \boldsymbol{N}$ *is the function defined by:* $\forall n.\ f_{\boldsymbol{K}}(r, n) = (l, (r, n)) \quad \wedge \quad \forall n.\ f_{\boldsymbol{K}}(l, (r, n)) = (r, n)$.

W : *In order to define* $\boldsymbol{W}$*, we need first to fix* $i, j \in \boldsymbol{N}$ *such that* $i \neq j$*. Then*
- $s_{\boldsymbol{W}} : ((\boldsymbol{N} \times \boldsymbol{N}) + ((\boldsymbol{N} + \boldsymbol{N}) + \boldsymbol{N})) + \boldsymbol{N} \rightharpoonup \boldsymbol{N}$ *is defined by* $s_{\boldsymbol{W}} = (p + (t + id_{\boldsymbol{N}})) + id_{\boldsymbol{N}}; (id_{\boldsymbol{N}} + t) + id_{\boldsymbol{N}}; t + id_{\boldsymbol{N}}; t$.
- $f_{\boldsymbol{W}} : ((\boldsymbol{N} \times \boldsymbol{N}) + ((\boldsymbol{N} + \boldsymbol{N}) + \boldsymbol{N})) + \boldsymbol{N} \rightharpoonup ((\boldsymbol{N} \times \boldsymbol{N}) + ((\boldsymbol{N} + \boldsymbol{N}) + \boldsymbol{N})) + \boldsymbol{N}$ *is the function defined by the following equations together with their symmetric closure:*

- $\forall n.\ f_{\boldsymbol{W}}(r,n) = (l,(r,(r,n)))$
- $\forall n.\ f_{\boldsymbol{W}}(l,(r,(l,(r,n)))) = (l,(l,(i,n)))$
- $\forall n.\ f_{\boldsymbol{W}}(l,(r,(l,(l,n)))) = (l,(l,(j,n)))$.

D : *In order to define* $\boldsymbol{D}$*, we need to fix* $i \in \boldsymbol{N}$*. Then*
- $s_{\boldsymbol{D}} : (\boldsymbol{N}\times\boldsymbol{N})+\boldsymbol{N} \rightharpoonup \boldsymbol{N}$ *is defined by* $s_{\boldsymbol{D}} = p + id_{\boldsymbol{N}}; t$.
- $f_{\boldsymbol{D}} : (\boldsymbol{N}\times\boldsymbol{N})+\boldsymbol{N} \rightharpoonup (\boldsymbol{N}\times\boldsymbol{N})+\boldsymbol{N}$ *is the function defined by:*
  $\forall n.\ f_{\boldsymbol{D}}(r,n) = (l,(i,n)) \quad \wedge \quad \forall n.\ f_{\boldsymbol{D}}(l,(i,n)) = (r,n)$.

δ : *In order to define* δ*, we need to fix* $i,j \in \boldsymbol{N}$*. Then*
- $s_\delta : (\boldsymbol{N}\times(\boldsymbol{N}\times\boldsymbol{N}))+\boldsymbol{N} \rightharpoonup \boldsymbol{N}$ *is defined by*
  $s_\delta = (id_{\boldsymbol{N}}\times p) + id_{\boldsymbol{N}}; p + id_{\boldsymbol{N}}; t$.
- $f_\delta : (\boldsymbol{N}\times(\boldsymbol{N}\times\boldsymbol{N}))+\boldsymbol{N} \rightharpoonup (\boldsymbol{N}\times(\boldsymbol{N}\times\boldsymbol{N}))+\boldsymbol{N}$ *is the function defined by:*
  $\forall n.\ f_\delta(r,n) = (l,(i,(j,n))) \quad \wedge \quad \forall n.\ f_\delta(l,(i,(j,n))) = (r,n)$.

F : *In order to define* $\boldsymbol{F}$*, we need to fix* $i,j \in \boldsymbol{N}$*. Then*
- $s_{\boldsymbol{F}} : ((\boldsymbol{N}\times\boldsymbol{N})+\boldsymbol{N}\times(\boldsymbol{N}+\boldsymbol{N}))+\boldsymbol{N} \rightharpoonup \boldsymbol{N}$ *is defined by*
  $s_{\boldsymbol{F}} = (p+(id_{\boldsymbol{N}}\times t))+id_{\boldsymbol{N}}; (id_{\boldsymbol{N}}+p)+id_{\boldsymbol{N}}; t$.
- $f_{\boldsymbol{F}} : ((\boldsymbol{N}\times\boldsymbol{N})+\boldsymbol{N}\times(\boldsymbol{N}+\boldsymbol{N}))+\boldsymbol{N} \rightharpoonup ((\boldsymbol{N}\times\boldsymbol{N})+\boldsymbol{N}\times(\boldsymbol{N}+\boldsymbol{N}))+\boldsymbol{N}$ *is the function defined by the following equations together with their symmetric closure:*
  - $\forall n.\ f_{\boldsymbol{F}}(r,n) = (l,(r,(i,(r,n))))$
  - $\forall n.\ f_{\boldsymbol{F}}(l,(r,(i,(l,n)))) = (l,(l,(j,n)))$.

There is a simple, intuitive, geometrical explanation of these combinators, which makes use of the language of boxes and wires. For example, let us consider the identity combinator $\boldsymbol{I}$. Since $\boldsymbol{I}$ has to satisfy the equation $\boldsymbol{I}x = x$, in order to define $\boldsymbol{I}$, it is convenient to regard $\boldsymbol{I}$ as a two-input/two-output function, up-to-coding. The Identity combinator just copies informations from the left-hand input-wire to the right-hand output-wire, and vice versa from the right-hand input-wire to the left-hand output-wire (see Fig. 2(i)). The fact that $\boldsymbol{I}$ satisfies the identity equation has a simple geometrical explanation. Let us apply $\boldsymbol{I}$ to a partial function x (see Fig. 2(ii)). Now *yank* the string connecting the input and the output wires of the result of the application, forgetting about the box corresponding to $\boldsymbol{I}$. This gives us immediately the expected result (see Fig. 2(iii)). Our argument is based on the *Yanking Property* of the trace on the symmetric monoidal category *Pfn* underlying our combinatory algebra. In particular, *Yanking* is one of the axioms characterizing the trace operation in the setting of traced symmetric monoidal categories.

Let us now consider the combinator $\boldsymbol{B}$ which satisfies the equation $\boldsymbol{B}xyz = x(yz)$. Concretely, the box for $\boldsymbol{B}$ has two input (and two output) wires for x and two input (and two output) wires for y, since both x and y are applied to an argument, one input (and one output) wire for z, which appears only as argument, plus one extra input (and one output) wire, along which the input-token (output-token) is intended to enter (exit). The connections of the wires inside the box for $\boldsymbol{B}$ are determined by the control flow between x, y, z in the right-hand part of the equation. First of all, the control flow passes from the input port of $\boldsymbol{B}$ to the input port of x. The second port of x is then connected

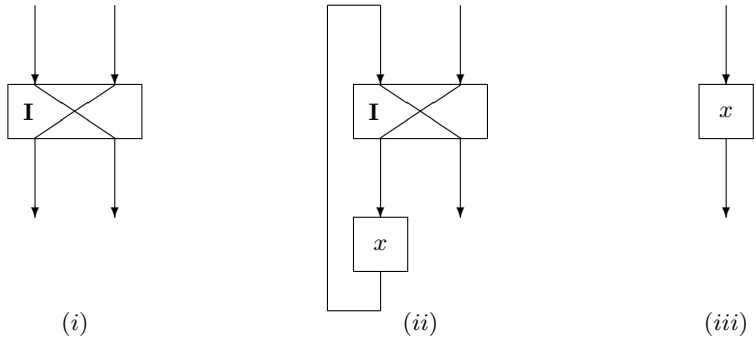


Fig. 2. Geometrical description of **I**.

to the input port of y, while the second port of y is connected to the unique port of z. The remaining connections are then obtained by symmetry (see Fig. 3(i)). Using the Yanking Property, one can then check that the result of the application of $\boldsymbol{B}$ to x, y, z is the expected one.

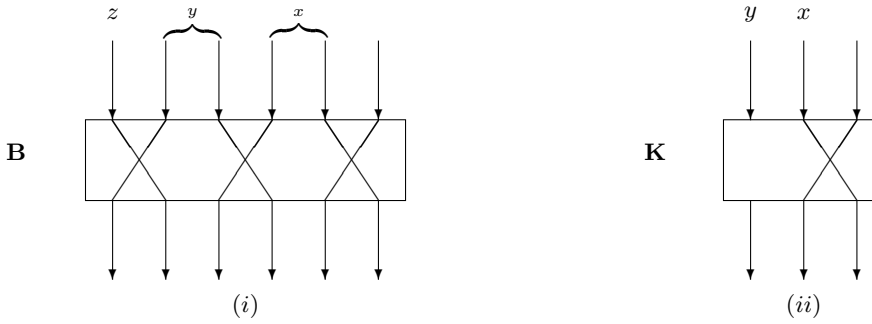
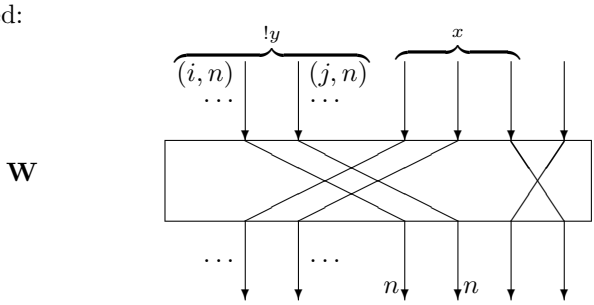


Fig. 3. Geometrical representations of **B**, **K**.

Now we briefly discuss the remaining combinators. The combinator $\boldsymbol{C}$ can be explained in a similar way as $\boldsymbol{B}$. The affine combinator $\boldsymbol{K}$ simply forgets about its second argument y (see Fig. 3(ii)).

In order to define $\boldsymbol{W}$, we need to fix two different indices $i, j \in \mathbf{N}$, tagging the copies of y which are used as arguments by x. The remaining copies of y are ignored:



The behavior of $\boldsymbol{D}, \delta, \boldsymbol{F}$ can be explained similarly (see [AL99a]).

Essentially, all the combinators of Definition 8 are functions that mediate the required interactions between the arguments simply by copying information between the various ports.

There are many possible conditions that can be imposed on partial functions in order to cut down the space $[\mathbf{N} \rightharpoonup \mathbf{N}]$, still maintaining closure under the application, !, and all the affine combinators. The subalgebra which gives rise to the fully-complete model of Section 5 is obtained by considering *partial involutions*:

Definition 9. *Let $f : \boldsymbol{N} \rightharpoonup \boldsymbol{N}$. f is a* partial involution *iff its graph is a symmetric relation. Let us denote by $[\boldsymbol{N} \rightharpoonup_{\mathsf{Inv}} \boldsymbol{N}]$ the space of partial involutions from $\boldsymbol{N}$ to $\boldsymbol{N}$.*

One can check that partial involutions are closed under the application, the !-operation, and all the combinators of Definition 8, i.e.:

Proposition 3. *$\mathcal{A}_{\mathsf{PInv}} = ([\boldsymbol{N} \rightharpoonup_{\mathsf{Inv}} \boldsymbol{N}], \bullet, !)$ is an affine combinatory algebra.*

$\mathcal{A}_{\mathsf{PInv}}$ is a highly constrained algebra, in which all computations are *reversible*. Partial involutions are reminiscent of the *copy-cat* strategies of game categories, in that the only computational effect that they have is that of *copying* informations from input to output wires.

5 A Fully Complete PER Model

In this section, we sketch the proof that the PER category over the LCA $\mathcal{A}_{\mathsf{PInv}}$ of Section 4 satisfies the Axioms of [AL99a], and hence it gives rise to a fully and faithfully complete PER model for ML-types.

The axiomatization of [AL99a] consists of two main steps. The first is an axiomatization of the fact that every morphism $f : 1 \to [\![T]\!]$, where T is an ML-type generates, under *decomposition*, a possibly *infinite* typed Böhm tree. The second step consists of an axiom which rules out infinite trees from the model is introduced. We start by discussing briefly the axioms for the decomposition. First of all, notice that the axiom which expresses the fact that the type $\forall \boldsymbol{X}.X_k$ is empty, and the *Uniformity of Threads* Axiom hold immediately on PER models. In fact, for the first axiom to hold, we need only to verify that the PER $\bigcap_{\boldsymbol{X}} X_k$ is the empty PER. This follows immediately, by instantiating X_k with the empty PER. The Uniformity of Threads Axiom follows from the isomorphism $\bigcap_{\boldsymbol{X}} ! \, \mathcal{R} \multimap ! S \simeq \bigcap_{\boldsymbol{X}} ! \, \mathcal{R} \multimap S$, which is an immediate consequence of Lemma 2 of Section 3. The proof of the validity of the remaining axioms for the Decomposition Theorem is based essentially on the nature of partial involutions, and it requires a careful analysis of their applicative behavior. The details of the lengthy proof appear in [AL99a]. The most difficult part of the proof of full completeness for the model $\mathrm{PER}_{\mathcal{A}_{\mathsf{PInv}}}$ consists in proving the *Finiteness Axiom*, i.e. in ruling out infinite typed trees. In particular, we prove that the trees generated by elements of PERs which are denotations of ML-types, via repeated applications of the

Decomposition Theorem, have *finite* height. In order to prove this finiteness result, we need to study an intermediate model, which contains also *approximant* terms of possibly infinite trees. This intermediate model consists of the hierarchy of *simple PERs* over the *Sierpinski PER*. This hierarchy gives rise to a model for the simply typed calculus with $\bot, \top$ constants at the base type. First of all, we prove an *Approximation Lemma* (along the lines of [AJM96]), which says that the graph of every partial involution f in a closed polymorphic PER can be viewed as the union of all its approximants. The approximants of f correspond, essentially, to the finite trees obtained by truncating at level k the tree generated from f by applying the Decomposition Theorem. Then, reasoning by contradiction, using the Typed Separability result of Section 1, and the fact that $\bot$ does not live in closed polymorphic PERs, we conclude that only trees with finite height belong to such PERs. The details of the proof appear in [AL99a].

6 Final Remarks and Directions for Future Work

We conclude this paper with a list of remarks and interesting issues which still remain to be addressed (some of them are currently under investigation).

• In this paper, we have presented a fully-complete model for ML-types. A natural question arises: what happens beyond ML-types. Here is a partial answer. Already at the type Nat $\to$ Nat, where *Nat* is the type of Church's numerals, i.e. $\forall X.(X \to X) \to X \to X$, the PER model of partial involutions is not fully-complete. In fact, not only all recursive functions, but even *all* functions from natural numbers to natural numbers, can be encoded in the type Nat $\to$ Nat. A similar problem arises even if we consider the term combinatory algebra. PER models as they are defined in this paper, do not seem to give full-completeness beyond ML-types. An innovative construction is called for here.

• Another question which arises naturally is whether the PER model over the linear term combinatory algebra is fully-complete at ML-types. We conjecture that this is the case, but a proof of this fact seems difficult. A logical relation technique relating the term algebra and the term subalgebra of partial involutions could be useful here. The interest of linear term algebras lies in the fact that the PER model generated by these is essentially the PER model shown to be fully-complete at algebraic types in [HRR90].

• We have presented a linear realizability technique for building PER categories over an LCA. These PER categories turn out to be linear categories. It would be interesting to carry on the investigation of the general properties of these categories, e.g. define coproducts, products, etc..

• Models of partial involutions are worthwhile investigating also for typed/untyped λ-calculi different from system F. E.g. strategies in the [AJM96] style, which are represented by partial involutions from Opponent moves to Player moves, should provide fully-complete models for simply typed λ-calculus with $\bot, \top$-base constants. In the untyped setting, partial involutions strategies could possibly provide fully-abstract models, alternative to those in [DFH99,KNO99].

• In the category $\mathrm{PER}_{\mathtt{PInv}}$, models of typed Böhm trees naturally arise (e.g. the

model induced by the Sierpinski PER). These are in particular models of the simply typed λ-calculus together with a *fixed point combinator*, as suggested by Alex Simpson. All these "infinite" calculi seem interesting by themselves, but have not yet been property investigated.

References

Abr96. S.Abramsky. Retracing some paths in Process Algebra, *Concur'96*, 1996.

Abr97. S.Abramsky. Axioms for Full Abstraction and Full Completeness, 1997, to appear.

Abr97a. S.Abramsky. Interaction, Combinators, and Complexity, Notes, Siena (Italy), 1997.

AHPS98. S.Abramsky, E.Haghverdi, P.Panangaden, P.Scott. Geometry of Interaction and Models of Combinatory Logic, 1998, to appear.

AJ94. S.Abramsky, R.Jagadeesan. New foundations for the Geometry of Interaction, *Inf. and Comp.* **111**(1), 1994, 53–119.

AJ94a. S.Abramsky, R.Jagadeesan. Games and Full Completeness for Multiplicative Linear Logic, *J. of Symbolic Logic* **59**(2), 1994, 543–574.

AJM96. S.Abramsky, R.Jagadeesan, P.Malacaria. Full Abstraction for PCF, 1996, Inf. and Comp. to appear.

AL99. S.Abramsky, J.Longley. Realizability models based on history-free strategies, Draft paper, 1999.

AL99a. S.Abramsky, M.Lenisa. Fully Complete Models for ML Polymorphic Types, Technical Report **ECS-LFCS-99-414**, LFCS, 1999 (available at http://www.dimi.uniud.it/~lenisa/Papers/Soft-copy-ps/lfcs99.ps.gz).

AM97. S.Abramsky, G.McCusker. Full abstraction for idealized Algol with passive expressions, *TCS* **227**, 1999, 3–42.

AM99. S.Abramsky, P.Mellies. *Concurrent Games and Full Completeness*, *LICS'99*.

AL91. A.Asperti, G.Longo. *Categories, Types and Structures*, Foundations of Computing Series, The MIT Press, 1991.

BC88. V.Breazu-Tannen, T.Coquand. Extensional models for polymorphism, *TCS* **59**, 1988, 85–114.

Cro93. R.Crole, *Categories for Types*, Cambridge University Press, 1993.

DFH99. P.Di Gianantonio, G.Franco, F.Honsell. Game Semantics for Untyped λ-calculus, *TLCA'99*, LNCS, 1999.

Gir72. J.Y.Girard. Interprétation functionelle et élimunation des coupures de l'arithmètique d'ordre supérieur, Thèse d'Etat, Université Paris VII, 1972.

Gir89. J.Y.Girard. Towards a Geometry of Interaction, *Contemporary Mathematics* **92**, 1989, 69–108.

Hug99. D.Hughes. *Hypergame Semantics: Full Completeness for System F*, D.Phil. thesis, University of Oxford, submitted 1999.

HRR90. J.Hyland, E.Robinson, G.Rosolini. Algebraic types in PER models, *MFPS*, M.Main *et al.* eds, LNCS **442**, 1990, 333–350.

HO96. M.Hyland, L.Ong. On full abstraction for PCF, *Inf. and Comp.*, 1996, to appear.

KNO99. A.Ker, H.Nickau, L.Ong. More Universal Game Models of Untyped λ-Calculus: The Böhm Tree Strikes Back, *CSL'99*, LNCS, 1999.

JSV96. A.Joyal, R.Street, D.Verity. Traced monoidal categories, *Math. Proc. Comb. Phil. Soc.* **119**, 1996, 447–468.

Lai97. J.Laird. Full abstraction for functional languages with control, *LICS'97*.
McC96. G.McCusker. Games and full abstraction for FPC, *LICS'96*, 1996.
Nic94. H.Nickau. Hereditarily sequential functionals, Proc. of the Symposium *Logical Foundations for Computer Science*, LNCS **813**, 1994.
Sta88. R.Statman. λ-definable functionals and $\beta\eta$-conversion, *Arch. Math. Logik* **23**, 1983, 21–26.

Subtyping with Power Types*

David Aspinall
http://www.dcs.ed.ac.uk/home/da

LFCS, University of Edinburgh, U.K.

Abstract. This paper introduces a typed λ-calculus called λ_{Power}, a predicative reformulation of part of Cardelli's *power type* system. Power types integrate subtyping into the typing judgement, allowing bounded abstraction and bounded quantification over both types and terms. This gives a powerful and concise system of dependent types, but leads to difficulty in the meta-theory and semantics which has impeded the application of power types so far. Basic properties of λ_{Power} are proved here, and it is given a model definition using a form of applicative structures. A particular novelty is the auxiliary system for *rough typing*, which assigns simple types to terms in λ_{Power}. These "rough" types are used to prove strong normalization of the calculus and to structure models, allowing a novel form of containment semantics without a universal domain.

Keywords: type theory, subtyping, dependent types.

1 Introducing Power Types

Power types were introduced in a seminal paper by Cardelli [4]. The notion is that $\mathsf{Power}(A)$ is a type "whose elements are all of the subtypes of the type A,"

$$\frac{A\ \mathsf{type}}{\mathsf{Power}(A)\ \mathsf{type}}$$

In place of a separate definition of subtyping, a relation between types is induced by inhabitation of power types, so $A \leq B \ =_{\text{def}}\ A : \mathsf{Power}(B)$. The rules for power types are chosen to make this definition sensible. Cardelli called the three basic rules power introduction, elimination and subtyping:

$$\frac{A\ \mathsf{type}}{A : \mathsf{Power}(A)} \qquad \frac{M : A \qquad A : \mathsf{Power}(B)}{M : B} \qquad \frac{A : \mathsf{Power}(B)}{\mathsf{Power}(A) : \mathsf{Power}(\mathsf{Power}(B))}$$

The first rule makes the induced subtyping relation reflexive. The second rule is the characteristic subtyping rule of *subsumption*, which adds subtype polymorphism to the system. The third rule expresses monotonicity of the Power operator, and together with the second rule, it makes the induced subtyping relation transitive. Other rules capture the subtyping behaviour of type constructors.

* Summary version. The full version [2] is available from my web page, address above.

P. Clote and H. Schwichtenberg (Eds.): CSL 2000, LNCS 1862, pp. 156–171, 2000.

Cardelli meant his type system to be used for programming languages with object-oriented features. Power types can encode *bounded* type abstraction and quantification used in OOP with the usual λ-abstraction and dependent function space, defining $\Lambda\alpha \leq A.M \ =_{\text{def}} \ \lambda\alpha{:}\,\mathsf{Power}(A).\,M$ and $\forall\alpha \leq A.B \ =_{\text{def}} \ \Pi\alpha{:}\,\mathsf{Power}(A).\,B$. This is a simplification, since there is no need to add new constructs. (The work described here grew from a slightly different application: in ASL+ [1], subtyping models specification refinement, and $\Lambda X \leq SP.M$ is a parameterised specification which can be applied to any refinement of SP.) Unfortunately, Cardelli's full power type system is tricky to handle: it has impredicative polymorphism via the $\mathsf{Type} : \mathsf{Type}$ axiom along with other features, rendering it undecidable, inconsistent when viewed as a logic, and difficult to give a semantics to. Later work on Quest [6] used power *kinds* instead, where $\mathsf{Power}(A)$ does not enjoy the status of a type itself.

As far as I know, power types have not been studied extensively since Cardelli's work; this is perhaps the first in-depth study. First I define a calculus called λ_{Power} (Section 3). It is almost a fragment of Cardelli's system, except for a richer power introduction rule and an equality judgement. Then I give some brief examples (Section 2), before considering the meta-theory (Section 4) and a semantics (Section 6). The semantics and some of the meta-theory are based on *rough typing*, a way of assigning "rough" non-dependent types to λ_{Power} terms (Section 5). Finally, Section 7 summarizes.

2 Examples in λ_{Power}

As a calculus of functions, λ_{Power} is no more expressive than the simply-typed λ-calculus.[1] In contrast with Cardelli's system, it is predicative: we cannot write a function which operates on any type, so there is no System F style universal polymorphism. All type operators are parameterised on subtypes of a given type. Despite this, λ_{Power} can express complex typings, because of the powerful combination of dependent types and arbitrarily nested power types.

2.1 A Simple Programming Example

Suppose int is an atomic type and let Γ_{PERM} be the context:

$$\begin{aligned}
\mathsf{nat} &: \mathsf{Power}(\mathsf{int}),\\
\mathsf{Upto} &: \mathsf{nat} \to \mathsf{Power}(\mathsf{nat}),\\
\mathsf{Perm} &: \Pi n{:}\mathsf{nat}.\ \mathsf{Power}((\mathsf{Upto}\,n) \to (\mathsf{Upto}\,n))\\
\mathsf{Invperm} &: \Pi n{:}\mathsf{nat}.\,(\mathsf{Perm}\,n) \to (\mathsf{Perm}\,n)
\end{aligned}$$

Imagine that $\mathsf{Upto}\,n$ stands for the set $\{\, m \in \mathsf{nat} \ \mid \ m \leq n \,\}$, and $\mathsf{Perm}\,n$ is the set of permutations of $\{\,1,\ldots,n\,\}$, which is a subset of the set of functions from

[1] If M is typable in λ_{Power}, then the type-erasure of M can be assigned a simple type, treating Π and Power as families of constants.

$\mathsf{Upto}\, n$ to $\mathsf{Upto}\, n$. The function $\mathsf{Invperm}\, n\, p$ yields the inverse of the permutation p on such a set. Here is a function to apply the inverse of a permutation of $\{1, \ldots, n\}$ to a number in that range:

$$\mathsf{ApplyPerm} \;=_{\text{def}}\; \lambda n{:}\mathsf{nat}.\, \lambda p{:}\mathsf{Perm}\, n.\, \lambda m{:}\mathsf{Upto}\, n.\; \mathsf{Invperm}\, n\, p\, m$$

Using subsumption for $\mathsf{Invperm}\, n\, p$, we can get the expected typing:

$$\Gamma_{\text{PERM}} \triangleright \mathsf{ApplyPerm} : \Pi n{:}\mathsf{nat}.\, (\mathsf{Perm}\, n) \;\to\; (\mathsf{Upto}\, n) \;\to\; (\mathsf{Upto}\, n).$$

which reveals that $\mathsf{ApplyPerm}\, n\, f$ is in fact a function from $\mathsf{Upto}\, n$ to $\mathsf{Upto}\, n$.

2.2 Subtyping Type Operators and Families

Systems of *higher-order subtyping* extend subtyping to type-constructors. The prototypical one is $F^{\omega}_{\leq}$ [5], in which one can declare a type variable ranging over type operators, $F \leq (\lambda\beta \leq \mathsf{nat}.\, \mathsf{List}(\beta \times \beta))$. A system with dependent types instead of polymorphism is $\lambda\mathrm{P}_{\leq}$ [3], in which one can declare a variable ranging over type families, $G \leq (\lambda x{:}\mathsf{nat}.\, \mathsf{Vec}_{\mathsf{nat}}(5{*}x))$. In the first case, F ranges over constructors that map a subtype β of nat to a subtype of $\mathsf{List}(\beta \times \beta)$; in the second case G ranges over constructors that map an element x of nat to a subtype of the type of vectors of numbers with $5{*}x$ elements. Both systems have a pointwise rule for subtyping operators and a corresponding application rule:

$$\frac{\Gamma,\, \alpha : K \triangleright A \leq B}{\Gamma \triangleright \lambda\alpha{:}K.\, A \leq \lambda\alpha{:}K.\, B} \qquad (\textsc{sub}\text{-}\lambda)$$

$$\frac{\Gamma \triangleright H \leq J \qquad \Gamma \triangleright J\, C : K}{\Gamma \triangleright H\, C \leq J\, C} \qquad (\textsc{sub-app})$$

The second premise of (SUB-APP) ensures that the application $J\, C$ is well-typed; this implies that $H\, C$ is also well-typed. Here's an example using (SUB-APP):

$$\frac{\begin{array}{lcl} G\, n & \leq & (\lambda x{:}\mathsf{nat}.\, \mathsf{Vec}_{\mathsf{nat}}(5 * x))\, n \\ (\lambda x{:}\mathsf{nat}.\, \mathsf{Vec}_{\mathsf{nat}}(5 * x))\, n & \leq & \mathsf{Vec}_{\mathsf{nat}}(5 * n) \end{array}}{G\, n \quad \leq \quad \mathsf{Vec}_{\mathsf{nat}}(5 * n)}$$

(where $n : \mathsf{nat}$ in the context). This is derived using conversion and transitivity.

In λ_{Power}, there is no rule directly corresponding to (SUB-λ). Indeed it is impossible to prove anything with the form $\Gamma \triangleright \lambda\alpha{:}K.\, A : \mathsf{Power}(C)$. The rules above are hard to interpret semantically, because the interpretation of $\lambda\alpha{:}K.\, A : \mathsf{Power}(C)$ must be considered pointwise rather than as a subset inclusion, so the meaning of Power would depend on its context in a term.

Perhaps surprisingly, power types can express similar typings without the pointwise rules. Suppose that F is a subtype of a type-constructor H with domain K; this is like asking F to be an element of $\Pi\alpha{:}K.\, \mathsf{Power}(H\, \alpha)$, since each

application $F\,M$ must be a subtype of $H\,M$. This "η-like" expansion for Π-types works uniformly[2] and we can declare:

$$\begin{array}{lcl} F & : & \Pi\beta{:}\mathsf{Power}(\mathsf{nat}).\ \mathsf{Power}(\mathsf{List}(\beta \times \beta)) \\ G & : & \Pi x{:}\mathsf{nat}.\ \mathsf{Power}(\mathsf{Vec}_{\mathsf{nat}}\,(5 * x)) \end{array}$$

To derive $G\,n \leq \mathsf{Vec}_{\mathsf{nat}}(5 * n)$ we need only one use of ordinary application:

$$\frac{G : \Pi x{:}\mathsf{nat}.\ \mathsf{Power}(\mathsf{Vec}_{\mathsf{nat}}\,(5 * x)) \qquad n : \mathsf{nat}}{G\,n : \mathsf{Power}(\mathsf{Vec}_{\mathsf{nat}}\,(5 * n))}$$

Substitution in the application rule for dependent products takes place of conversion and transitivity needed before, so derivations in λ_{Power} can be more direct.[3]

2.3 λ_{Power} as a Logical Framework

λ_{Power} is related to λP, which underlies the Edinburgh LF [9]. It's quite easy to see that λ_{Power} can be used in the same way as λP. Let υ be an atomic type. Then declare a universe of types by writing $\mathsf{U} =_{\mathrm{def}} \mathsf{Power}(\upsilon)$. We can use U in place of Type in LF, to declare the term formers and judgements of a logic. If $\Gamma \rhd A : \mathsf{U}$ and $\Gamma, x : A \rhd B : \mathsf{U}$, then we do *not* have $\Gamma \rhd \Pi x{:}A.\,B : \mathsf{U}$, but rather $\Gamma \rhd \Pi x{:}A.\,B : \mathsf{Power}(\Pi x{:}A.\,\upsilon)$. Since λP lacks quantification or abstraction over types, this difference has little effect, and we can translate any λP judgement into one which holds in λ_{Power}.[4] With power types we can declare one syntactic category to be a subtype of another, or one judgement to be a subtype of another, so that every proof of the first judgement is also a proof of the second. This is also possible in the proposals studied in [11, 3], but λ_{Power} goes beyond both these systems by allowing refinements of the universe U itself.

Gardner proposed doing this [8] to help adequacy proofs. She defined a framework ELF+ which distinguishes between terms that represent: object-level syntax, proof terms, and other terms. To emulate ELF+ in λ_{Power}, declare three subtypes: $\mathtt{Sort} : \mathsf{Power}(\mathsf{U})$, $\mathtt{Judge} : \mathsf{Power}(\mathsf{U})$, and $\mathtt{Type} : \mathsf{Power}(\mathsf{U})$.

An encoding where power types are useful is higher-order logic (HOL). Simple types τ of the form ι, o, and $\tau \Rightarrow \tau$ are encoded in an LF type $\mathtt{dom} : \mathtt{Type}$, with $\mathtt{i}, \mathtt{o} : \mathtt{dom}$, $\Rightarrow : \mathtt{dom} \to \mathtt{dom} \to \mathtt{dom}$ and $\mathtt{obj} : \mathtt{dom} \to \mathtt{Type}$. HOL terms with domain τ are represented as elements of $\mathtt{obj}(\tau)$. ELF+ improves this, showing $\mathtt{dom}$ and $\mathtt{obj}$ to be artifacts of the encoding, inhabiting $\mathtt{Type}$, and typing $\mathtt{obj} : \mathtt{dom} \to \mathtt{Sort}$, showing that elements of $\mathtt{obj}(\tau)$ correspond to object logic syntax. But in both LF and ELF+, the proliferation of $\mathtt{obj}$ quickly pollutes large terms. In λ_{Power}, we can remove it altogether and declare $\mathtt{dom} : \mathsf{Power}(\mathtt{Sort})$. The mapping $\mathtt{obj}$ is now implicit; the representation of the logic becomes more concise, yet no less accurate. For example, the application term former becomes:

$$\mathtt{app} : \Pi s, t{:}\mathtt{dom}.\,(s \Rightarrow t) \to s \to t$$

[2] This idea also appears in Crary's λK system which has power kinds [7].
[3] But *practical* effects on type-checking algorithms have not been investigated yet.
[4] Perhaps, moreover, λ_{Power} is conservative over λP under this translation.

instead of

$$\mathtt{app} : \Pi s, t{:}\,\mathtt{dom}.\,\mathtt{obj}(s \Rightarrow t) \to \mathtt{obj}(s) \to \mathtt{obj}(t).$$

Although simple, it is important to emphasise that this example goes beyond many other subtyping proposals. Power types apply uniformly; other systems would have to be extended with sub-kinding to cope with this example.

3 The System λ_{Power}

Let $\mathbf{V}$ be a fixed countable infinite set of variables and $\mathcal{K}$ be a set of atomic type constants. The set $\mathbf{T}_{\mathcal{K}}$ of *pre-terms* is given by:

$$\mathbf{T} ::= \mathcal{K} \quad | \quad \mathbf{V} \quad | \quad \lambda \mathbf{V}{:}\mathbf{T}.\,\mathbf{T} \quad | \quad \mathbf{T}\,\mathbf{T} \quad | \quad \Pi \mathbf{V}{:}\mathbf{T}.\,\mathbf{T} \quad | \quad \mathit{Power}(\mathbf{T})$$

(writing $\mathbf{T}$ as short for $\mathbf{T}_{\mathcal{K}}$). For meta-variables I use $x, y, \ldots \in \mathbf{V}$, $\kappa, \ldots \in \mathcal{K}$, and $A, B, \ldots, M, N, \ldots \in \mathbf{T}$. Usual conventions are used for writing pre-terms. A *pre-context* is a sequence of variable declarations $x_1 : A_1, x_2 : A_2 \ldots$ where no variable is declared more than once. The empty pre-context is sometimes written $\langle\rangle$, otherwise it is invisible. I use Γ and variants to range over pre-contexts.

Not all pre-terms make sense. The well-formed pre-terms consist of *terms* and *types*, defined in Definition 3.1 below. These are not disjoint; types are also terms of the calculus. Terms and types are defined via three judgement forms:

$\triangleright\, \Gamma$	Γ is a well-formed context
$\Gamma \triangleright M : A$	In context Γ, M has type A
$\Gamma \triangleright M = N : A$	In context Γ, M and N are equal at type A

These judgements are defined simultaneously by the rules shown at the end of the paper. The system λ_{Power} is close to a predicative fragment of Cardeilli's original system [4]; the difference is that we use an equality judgement in the presentation, and the more powerful (REFL). Here is a brief outline of the rules.
Context formation (Figure 2). These rules are standard. The judgement $\Gamma \triangleright A : \mathit{Power}(B)$ serves to say that A is a well-formed type, as well as asserting that A is a subtype of B. This is a general pattern.
Typing rules (Figure 1). Most rules are standard. The rule (ATOMIC) introduces atomic types; each atomic type is a subtype of itself, so is self-evidently well-formed. The rule-scheme (REFL) is novel, it expands to this:

$$\frac{\Gamma \triangleright M : \Pi x_1{:}A_1.\ \ldots \Pi x_n{:}A_n.\ \mathit{Power}(B)}{\Gamma \triangleright M : \Pi x_1{:}A_1.\ \ldots \Pi x_n{:}A_n.\ \mathit{Power}(M\, x_1 \cdots x_n)}$$

Reflexivity of subtyping for types is the case that $n = 0$. For each $n > 0$, the rule (REFL) asserts reflexivity of subtyping for n-ary type-valued functions[5] (the example in Section 2.2 motivates this). The rule (Π) generalises the usual

[5] A technical note: (REFL) adds a case of η-subject reduction to the sytem; if $y : \Pi x{:}A.\ \mathit{Power}(B)$ then with (λ) we get $\lambda x{:}A.\, y\, x : \Pi x{:}A.\ \mathit{Power}(y\, x)$, but we need (REFL) to get $y : \Pi x{:}A.\ \mathit{Power}(y\, x)$.

contravariant subtyping rule for function spaces to dependent products. The last premise is a well-formedness check.
Equality rules (Figure 2). These rules are standard.

Definition 3.1 (Terms, types and subtypes). *We say that* M *is a* Γ-term *if for some* A, $\Gamma \triangleright M : A$, A *is a* Γ-type *if for some* B, $\Gamma \triangleright A : \mathsf{Power}(B)$, *and* A *is a subtype* of B in Γ *if* $\Gamma \triangleright A : \mathsf{Power}(B)$.

The adjective "well-formed" emphasises that a pre-term can be typed in the calculus, as required by Definition 3.1. There are three derived judgement forms:

$$\begin{array}{lll} \Gamma \triangleright A \leq B & =_{\text{def}} & \Gamma \triangleright A : \mathsf{Power}(B) \\ \Gamma \triangleright A\ \mathsf{type} & =_{\text{def}} & \text{for some } B, \quad \Gamma \triangleright A : \mathsf{Power}(B) \\ \Gamma \triangleright A = B & =_{\text{def}} & \text{for some } C, \quad \Gamma \triangleright A = B : \mathsf{Power}(C) \end{array}$$

Section 4 shows that these definitions make sense.

4 Properties of λ_{Power}

The development begins with showing derivability of several rules: that the induced subtype relation is a pre-order, and that type equality is reflexive and symmetric. I distinguish derivable rules from those which are admissible but not derivable because in the semantics we consider some important admissible rules (namely, substitution and thinning) as part of the system, making sure they are valid in every model. Some authors add these "important" admissible rules to the presentation but this spoils the inductive proof of several meta-properties.

Notation 4.1. Let $\Gamma \equiv x_1 : A_1, \ldots$ be a pre-context. Let $Dom(\Gamma) =_{\text{def}} \{x_1, \ldots\}$ be the set of variables Γ declares, $\Gamma|_{x_i} =_{\text{def}} x_1 : A_1, \ldots, x_{i-1} : A_{i-1}$ be the restriction of Γ up to x_{i-1}. Define $\Gamma(x_i) =_{\text{def}} A_i$, viewing Γ as a partial mapping $\Gamma : \mathbf{V} \rightharpoonup \mathbf{T}$. Define $\Gamma \subseteq \Gamma'$ iff every declaration $x_i : A_i$ in Γ also appears in Γ'.

I use J to range over judgements of the system, and $\Gamma \triangleright J$ for a judgement with context Γ. A *simultaneous substitution* is a partial map from variables to pre-terms; a *renaming* is the special case of a simultaneous substitution which is a bijection on a subset of $\mathbf{V}$. Substitution is extended to contexts and judgements componentwise, *e.g.,* if $\Gamma \equiv x_1 : A_1, \ldots$ then $\Gamma[N/x] \equiv x_1 : A_1[N/x], x_2 :, \ldots$.

We first prove by induction on derivations that the usual good properties for subtyping systems hold: context formation, renaming, thinning, substition and bound narrowing (replacing $x : A$ with $x : A'$ where $A' : \mathsf{Power}(A)$). Next we show the important formation and type correctness properties.

Proposition 4.2 (Formation).

1. $\Gamma \triangleright \lambda x{:}A.\, M : C \implies \Gamma \triangleright A\ \mathsf{type}$ *and* $\exists B.\ \Gamma, x : A \triangleright M : B$.
2. $\Gamma \triangleright M\, N : C \implies \exists A, B.\ \Gamma \triangleright M : \Pi x{:}A.\, B$ *and* $\Gamma \triangleright N : A$.
3. $\Gamma \triangleright \Pi x{:}A.\, B : C \implies \Gamma \triangleright A\ \mathsf{type}$ *and* $\Gamma, x : A \triangleright B\ \mathsf{type}$.
4. $\Gamma \triangleright \mathsf{Power}(A) : C \implies \Gamma \triangleright A\ \mathsf{type}$.

Proposition 4.3 (Type correctness).

1. $\Gamma \triangleright M : A \implies \Gamma \triangleright A\ \mathit{type}$.
2. $\Gamma \triangleright M = N : A \implies \Gamma \triangleright A\ \mathit{type}$ *and* $\Gamma \triangleright M, N : A$.

The few basic equality rules of Figure 2 have some important admissible rules as consequences, proved using the propositions above. These include congruence rules for the type constructors, and rules of subsumption, conversion and substitution for the equality judgement itself. For details, see [2]. An important intermediate stage is proving the transitivity of type equality, using this rule:

$$\frac{\Gamma \triangleright A = B : \mathsf{Power}(C)}{\Gamma \triangleright A = B : \mathsf{Power}(B)} \quad (\textsc{eq-sub-refl})$$

This shows that type equality is "absolute", in the sense that the derivability of $A = B : \mathsf{Power}(C)$ is not affected by the choice of C when A and B are types such that $A, B : \mathsf{Power}(C)$.[6] In general, we expect this for type equality, but not necessarily for term equality. It is typical for subtyping calculi that the equality of two terms may vary across their common types. The semantics considered later reflects these ideas.

4.1 Further Properties

We would like to prove more about the λ_{Power} system than the properties in the previous section. One desirable property is the important practical property of subject reduction: If $\Gamma \triangleright M : A$ and $M \longrightarrow\!\!\!\!\!\rightarrow_{\beta\eta} M'$, then $\Gamma \triangleright M' : A$ too. Unfortunately it seems difficult to prove for λ_{Power}. The key is a *generation principle*, which gives a way of decomposing derivations by stating how a particular judgement was derived. Proposition 4.2 is a weak generation principle, but it is not strong enough. For a judgement $\Gamma \triangleright N : C$, we need a principle which connects C with the judgements about subterms of N asserted to exist. For the λ-case, a first approximation might be this: if $\Gamma \triangleright \lambda x{:}A.\, M : C$ then $C = \Pi x{:}A'.\, B'$, where $\Gamma \triangleright A' \leq A$ and there is a B such that $\Gamma, x : A \triangleright M : B$ and $\Gamma, x : A' \triangleright B \leq B'$. This captures the observation that after applying (λ) there can be several subsumptions and conversions through which $\Pi x{:}A.\, B$ mutates into C:

$$\dfrac{\dfrac{\dfrac{\dfrac{x : A \triangleright M : B}{\lambda x{:}A.\, M : \Pi x{:}A.\, B}}{\vdots}}{\lambda x{:}A.\, M : C_j} \qquad C_j \leq C_{j+1}}{\lambda x{:}A.\, M : C_{j+1}}\ (\textsc{sub})$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\lambda x{:}A.\, M : C_{j+1}}{\vdots}}{\lambda x{:}A.\, M : C_k} \qquad C_k = C_{k+1}}{\lambda x{:}A.\, M : C_{k+1}}\ (\textsc{conv})}{\vdots}}{\lambda x{:}A.\, M : C}$$

[6] If there is a C' such that $B : \mathsf{Power}(C')$ then $A = B : \mathsf{Power}(C')$ too by (EQ-SUB-REFL), (*Power*), and subsumption for equality.

The cut-like rules (SUB) and (CONV) make it hard to prove the statement directly, because to "join up" the arbitrary C_i's in the intervening typings we nant to use the generation principle being proved. It is worse than this, because (REFL) can introduce other detours, so the putative statement above needs altering.

The traditional syntactic solution to this problem is to give a *syntax-directed* reformulation of the system, eliminating the cut-like rules. Unfortunately this technique does not apply easily to λ_{Power}. The sticking point is bounded operator abstraction which makes it hard to prove substitution lemmas in the syntax-directed system before proving other properties which depend on substitution. A related solution involves giving a revised definition of the subtyping relation from the outset, on pre-terms. This too is difficult for power types, which have no separate subtyping judgement anyway. The problem remains open.

5 Rough Type-Checking

Although λ_{Power} is a dependently-typed calculus, we can approximate type-checking using "rough" types without term dependency. Rough type-checking is useful because it enforces a structural well-formedness property that is necessary for typability in the full system. Two pre-terms which are in the full typing relation of λ_{Power} have related rough types, and two terms which are equal in the equational theory have the same rough type. The idea of rough type-checking comes from [12], which suggested that rough types could be used to give a semantics to ASL+. This is done for λ_{Power} in Section 6. Another application of rough types is the proof of strong normalization for λ_{Power} [2].

5.1 Rough Typing System

Given a set $\mathcal{K}$ of atomic types, the set $\mathbf{Ty}_{\mathcal{K}}$ of *rough types over* $\mathcal{K}$ consists of type constants, arrow types, and power types, defined by the grammar:

$$\mathbf{Ty} ::= \mathcal{K} \quad | \quad \mathbf{Ty} \Rightarrow \mathbf{Ty} \quad | \quad P(\mathbf{Ty})$$

(writing $\mathbf{Ty}$ as short for $\mathbf{Ty}_{\mathcal{K}}$). I use $\tau, \upsilon, \ldots$ to range over $\mathbf{Ty}$. There are two rough typing judgements, using filled triangles:

$\blacktriangleright\ \Gamma$	Γ is a roughly-typable context
$\Gamma \blacktriangleright M : \tau$	M has rough type τ in Γ

The judgements are defined inductively by the rules in Figure 3. Notice that full λ_{Power} contexts are used in the rough typing judgements.

One can understand the rough typing rules as an abstract interpretation of terms-in-context, which follows set-theoretic intuitions for the calculus. The rough type of a term tells us what kind of beast it denotes: lambda terms denote functions and have arrow rough-types; atomic types and power types denote collections of values and have power rough-types. A term $\Pi x{:}A.\, B$ has a rough type of the form $P(\tau \Rightarrow \upsilon)$, indicating that it denotes a collection of functions.

Example 5.1. To illustrate rough typing, recall the example context Γ_{PERM} from Section 2.1. We can derive these rough typings:

$$\Gamma_{\mathrm{PERM}} \blacktriangleright \mathit{Perm} : \mathit{int} \Rightarrow \mathsf{P}(\mathit{int} \Rightarrow \mathit{int})$$

$$\Gamma_{\mathrm{PERM}} \blacktriangleright \mathit{Invperm} : \mathit{int} \Rightarrow (\mathit{int} \Rightarrow \mathit{int}) \Rightarrow (\mathit{int} \Rightarrow \mathit{int}).$$

At once we see how "rough" this is: *Perm* and *Invperm* were defined on *nat*, but *nat* gets replaced by the atomic type *int*.

In general, rough typing judgements — or to be more precise, their translation got by mapping $\tau \Rightarrow \upsilon$ to $\Pi x{:}\tau.\, \upsilon$ — do not hold in the full λ_{Power} type system. Certainly we do *not* have:

$$\Gamma_{\mathrm{PERM}} \triangleright \mathit{Perm} \;:\; \mathit{int} \to \mathit{Power}(\mathit{int} \to \mathit{int})$$

because, for starters, *Perm* is not defined on all of *int*. In Proposition 5.4, we prove that typability in the full calculus guarantees rough typability. The above example shows that the converse fails, since:

$$\Gamma_{\mathrm{PERM}}, i : \mathit{int} \blacktriangleright \mathit{Perm}\, i :\; \mathsf{P}(\mathit{int} \Rightarrow \mathit{int})$$

but $\mathit{Perm}\, i$ cannot be typed in the full system[7].

It is easier to establish properties of the rough-typing system than the full system, because the types are non-dependent and subtyping has been removed. First, we have the usual thinning, substitution and also strengthening properties for the rough type system. Then we can prove decidability and subject reduction.

Proposition 5.2 (Properties of rough typing).

1. *If* $\Gamma \blacktriangleright M : \tau$, *then* τ *is the unique such rough type.*
2. *Rough type-checking and rough type-inference are decidable*[8]

Proposition 5.3 (Subject reduction for rough typing). *If* $\Gamma \blacktriangleright M : \tau$ *and* $M \longrightarrow\!\!\!\!\!\rightarrow_{\beta\eta} M'$, *then* $\Gamma \blacktriangleright M' : \tau$ *too.*

The agreement property below is the important connection between rough types and typing in full λ_{Power}, claimed at the beginning of this section.

Theorem 5.4 (Agreement of rough typing).

1. *If* $\triangleright \Gamma$ *then* $\blacktriangleright \Gamma$.
2. *If* $\Gamma \triangleright M : A$ *then for some* $\tau \in \mathbf{Ty}$, $\Gamma \blacktriangleright M : \tau$ *and* $\Gamma \blacktriangleright A :\; \mathsf{P}(\tau)$.
3. *If* $\Gamma \triangleright M = N : A$ *then for some* $\tau \in \mathbf{Ty}$, $\Gamma \blacktriangleright M, N : \tau$ *and* $\Gamma \blacktriangleright A :\; \mathsf{P}(\tau)$.

[7] To prove this rigorously we need to use a generation principle or model construction.
[8] Assuming we can decide syntactic identity of atomic types, *i.e.,* whether $\kappa \equiv \kappa'$.

6 Semantics

Subtyping calculi have two basic kinds of model. With a typed value space, we may choose a *coercion semantics*, where each use of subsumption is modelled by the insertion of a *coercion* from type to supertype. If $A \leq B$, there is a map $c_{A,B} : [\![A]\!] \to [\![B]\!]$. This is a general setting, but it requires a *coherence* property of the interpretation, to show that different ways of putting coercions into a coercion-free judgement have the same interpretation. The coherence property can be difficult to establish. The other kind of model is a *containment semantics* in which subtyping is interpreted as containment between types: $[\![A]\!] \subseteq [\![B]\!]$. There is no problem of coherence in this case, but there is a difficulty with the rule for subtyping Π-types. In the syntax we have $\mathsf{int} \to \mathsf{int} \leq \mathsf{nat} \to \mathsf{int}$, but this does not hold as a set-theoretic inclusion; $\mathbf{Z} \to \mathbf{Z} \not\subseteq \mathbf{N} \to \mathbf{Z}$ when the semantic $\to$ is set-theoretic function space. This is usually solved by interpreting $\mathsf{nat} \to \mathsf{int}$ as the collection of all partial functions defined *at least* on $\mathbf{N}$; then the inclusion $\mathbf{Z} \rightharpoonup \mathbf{Z} \subseteq \mathbf{N} \rightharpoonup \mathbf{Z}$ holds. But then we need a universe of values over which to form this "collection of all partial functions," and this is what leads to an *untyped* value space in containment semantics. Typically, the untyped value space is the domain of a model of the untyped λ-calculus, and the denotation of a term is defined using its type-erasure [10]. But it is a surprising overkill to base a semantics for a calculus as simple as $\lambda_{\leq}$ (the extension of $\lambda^{\to}$ with subtyping) on a model of the untyped λ-calculus which requires a universal domain.

For power types, a containment semantics is natural and is the intended model for ASL+. I shall and give a containment semantics for λ_{Power} which is nevertheless based on a typed value space. Rough types make this possible. Whenever $A \leq B$, then A and B have the same rough type $\mathsf{P}(\tau)$, say, and so both may be interpreted as subsets of the interpretation of τ: $[\![A]\!] \subseteq [\![B]\!] \subseteq [\![\tau]\!]$. Since every type $\Pi x{:}C.\, D$ has a rough type of the form $\mathsf{P}(\tau_C \Rightarrow \tau_D)$, we can form the "collection of all functions with domain at least $[\![C]\!]$" using $[\![\tau_C]\!]$ as a universe, instead of a universal domain. The final ingredient is the equational theory of subtyping, where the equality of two terms may depend upon the type at which they are viewed. To deal with this, we use PERs rather than sets. The following sections give an abstract model definition for λ_{Power} based on these ideas, beginning from applicative structures. The reason for an abstract definition is to capture both the intended model and a term model; the term model is unusual for using an *external equality* notion rather than quotients (because of this extensionality is not assumed from the start). Space reasons prevent description of the term model here, see [2] for details.

6.1 Structures

A λ_{Power} applicative structure is similar to a typed-applicative structure for $\lambda^{\to}$ It provides semantic domains for every rough type; the domains are sets.

Definition 6.1 (λ_{Power} applicative structure). *A λ_{Power}* applicative structure $\mathcal{D} = \langle \mathcal{D}, \mathsf{Const}, \mathsf{App} \rangle$ *consists of a family of sets* $\{\, \mathcal{D}^{\tau} \,\}_{\tau \in \mathbf{Ty}}$*; a constant*

$\mathsf{Const}(\kappa) \in \mathcal{D}^{P(\kappa)}$ *for each* $\kappa \in \mathcal{K}$*, and a mapping* $\mathsf{App}^{\tau,\upsilon} : \mathcal{D}^{\tau \Rightarrow \upsilon} \to \mathcal{D}^{\tau} \to \mathcal{D}^{\upsilon}$ *for each* $\tau, \upsilon \in \mathbf{Ty}$*. Type annotations* τ, υ *are sometimes omitted for brevity.* □

Notation 6.2. Given a set S, $\mathrm{REL}(S)$ is the set of relations on S, $\mathrm{REL}(S) =_{\mathrm{def}} Pow(S \times S)$. If $R \in \mathrm{REL}(S)$, then $dom(R) = \{\, a \mid a\ R\ a \,\}$. A relation is a *partial equivalence* (PER) if it is symmetric and transitive; $\mathrm{PER}(S)$ is the set of PERs on S. The notation $a \mapsto f(a)$ stands for the function mapping a to $f(a)$.

Example 6.3 (Full hierarchy structure). Given a family of sets and PERs $C = \{\, C_\kappa, R_\kappa \in \mathrm{PER}(C_\kappa) \,\}_{\kappa \in \mathcal{K}}$, the *full hierarchy* $\mathcal{F}_C$ on C has $\mathcal{F}^{\kappa} = C_\kappa$, $\mathcal{F}^{\tau \Rightarrow \upsilon} = \mathcal{F}^{\tau} \to \mathcal{F}^{\upsilon}$, $\mathcal{F}^{P(\tau)} = \mathrm{REL}(\mathcal{F}^{\tau})$, $\mathsf{App}(f, m) = f(m)$, and $\mathsf{Const}(\kappa) = R_\kappa$. □

In the full hierarchy structure, $\mathcal{F}^{P(\tau)}$ is the set of all relations over $\mathcal{F}^{\tau}$, rather than the set of all PERs. This is for a technical reason: because the interpretation in the full structure (Example 6.7) is defined over rough types, the type-constructors are not guaranteed to construct PERs.

6.2 Environments and Interpretations

For each roughly-typable context Γ, we define a semantic domain $\mathcal{D}^{\Gamma}$ by induction on Γ, setting $\mathcal{D}^{\langle\rangle} = \{\star\}$ and $\mathcal{D}^{\Gamma, x:A} = \mathcal{D}^{\Gamma} \times \mathcal{D}^{\tau}$, where $\Gamma \blacktriangleright A : P(\tau)$ and $\{\star\}$ is some singleton set. A *Γ-environment* is a nested tuple $\eta \in \mathcal{D}^{\Gamma}$. Because we use a name-free denotation, if Φ is a renaming on $Dom(\Gamma)$ then η is a $\Phi(\Gamma)$-environment iff it is a Γ-environment. Given a Γ-environment $\eta \in \mathcal{D}^{\Gamma}$, we can define a projection function from the variables of Γ:

$$\eta^{\langle\rangle}(y) \qquad \text{undefined, for all } y.$$
$$\eta^{\Gamma, x:A}(y) = \begin{cases} snd(\eta), & \text{if } y \equiv x, \\ (fst\,\eta)^{\Gamma}(y) & \text{if } y \not\equiv x. \end{cases}$$

So if $\Gamma|_x \blacktriangleright \Gamma(x) : P(\tau)$, then $\eta^{\Gamma}(x) \in \mathcal{D}^{\tau}$. Thinning between environments is defined using this projection notation. If $\Gamma_1 \subseteq \Gamma_2$, $\eta_1 \in \mathcal{D}^{\Gamma_1}$ and $\eta_2 \in \mathcal{D}^{\Gamma_2}$, then ${\eta_1}^{\Gamma_1} \subseteq {\eta_2}^{\Gamma_2}$ iff $\eta_1^{\Gamma_1}(x) = \eta_2^{\Gamma_2}(x)$ for all $x \in Dom(\Gamma_1)$. The notation $\eta|_{x_i}$ stands for the restriction of a Γ-environment η to variables declared before x_i, meaning the shorter tuple $fst^{n-i}(\eta)$ where x_i is the ith variable of n declared in Γ.

Unlike a partial function environment, this tupled form has an explicit notion of the domain $\mathcal{D}^{\Gamma}$ associated to a context. We need this because relations over $\mathcal{D}^{\Gamma}$ are used in the soundness proof. Using tuples gives us an interpretation function reminiscent of the semantics of $\lambda^{\to}$ in (set-like) CCCs.

Definition 6.4 (λ_{Power} interpretation). *A* λ_{Power} interpretation *in* $\mathcal{D}$ *consists of a meaning function* $[\![\Gamma \blacktriangleright - : \tau]\!]_{-} : \mathbf{T} \rightharpoonup \mathcal{D}^{\Gamma} \to \mathcal{D}^{\tau}$*, for each roughly-typable context* Γ *and* $\tau \in \mathbf{Ty}$*, such that whenever* $\Gamma \blacktriangleright M : \tau$ *and* $\eta \in \mathcal{D}^{\Gamma}$*, then* $[\![\Gamma \blacktriangleright M : \tau]\!]_{\eta} \in \mathcal{D}^{\tau}$*, and for each* $\tau \in \mathbf{Ty}$*, a mapping* $\mathsf{Rel}^{\tau} : \mathcal{D}^{P(\tau)} \to \mathrm{REL}(\mathcal{D}^{\tau})$ *Type annotations* τ *may be omitted for brevity.* □

When $a \in \mathcal{D}^{P(\tau)}$, I sometimes use R_a as shorthand for $\mathit{Rel}^{\tau}(a)$. The mapping Rel models the behaviour (or *extension*) of elements denoting types, just as App models the extension of elements denoting functions. It is part of the interpretation so we can consider different "views" of types in the same structure. Definition 6.4 does not require *a priori* that R_a is a PER, for the reason outlined before; instead the soundness theorem will imply that any type of λ_{Power} denotes a PER. This differs slightly from other model definitions for dependent types which use a *partial definition*, proved to be total on well-typed terms. Instead we require that an interpretation is defined on all *roughly-typed* terms.

6.3 Models

We will use some constructions on relations. Let $\mathcal{D}$ be a λ_{Power} interpretation. Given $R \in \mathrm{REL}(\mathcal{D}^{\tau})$ and $G \in \mathit{dom}(R) \to \mathrm{REL}(\mathcal{D}^{\upsilon})$, we define $\Pi(R, G) \in \mathrm{REL}(\mathcal{D}^{\tau \Rightarrow \upsilon})$, $\mathcal{P}\mathrm{wr}(R) \in \mathrm{PER}(\mathcal{D}^{P(\tau)})$ by:

$$\begin{array}{lll} f\ \ \Pi(R,G)\ \ g & \text{iff} & \forall a, b.\ (a\ \ R\ \ b) \Longrightarrow \mathit{App}(f,a)\ \ G(a)\ \ \mathit{App}(g,b). \\ a\ \ \mathcal{P}\mathrm{wr}(R)\ \ b & \text{iff} & \mathit{Rel}(a) = \mathit{Rel}(b) \in \mathrm{PER}(\mathcal{D}^{\tau}) \quad \text{and} \quad \mathit{Rel}(a) \subseteq R. \end{array}$$

Fact 6.5. *If $R \in \mathrm{PER}(\mathcal{D}^{\tau})$ and $G(a) = G(b) \in \mathrm{PER}(\mathcal{D}^{\upsilon})$ whenever $a\ \ R\ \ b$, then $\Pi(R, G) \in \mathrm{PER}(\mathcal{D}^{\tau \Rightarrow \upsilon})$.*

Definition 6.6 (λ_{Power} environment model). *A λ_{Power} environment model for a structure $\mathcal{D}$ is an interpretation for $\mathcal{D}$ such that the following 9 conditions are satisfied, for all suitable roughly-typable terms. For roughly-typable contexts $\Gamma, \Gamma_1, \Gamma_2$ and all $\eta \in \mathcal{D}^{\Gamma}$, $\eta_1 \in \mathcal{D}^{\Gamma_1}$, $\eta_2 \in \mathcal{D}^{\Gamma_2}$ with ${\eta_1}^{\Gamma_1} \subseteq {\eta_2}^{\Gamma_2}$,*

$$\begin{array}{llllll} \mathsf{CONST} & [\![\kappa]\!]_\eta = \mathit{Const}(\kappa). & \mathsf{VAR} & [\![x]\!]_\eta = \eta^{\Gamma}(x). \\ \mathsf{CONST2} & R_{[\![\kappa]\!]_\eta} \in \mathrm{PER}(\mathcal{D}^{\kappa}). & \mathsf{APP} & [\![M\,N]\!]_\eta = \mathit{App}([\![M]\!]_\eta, [\![N]\!]_\eta). \end{array}$$

FAMILY *If for all a, b $(a\ \ R_{[\![A]\!]_\eta}\ \ b) \Longrightarrow R_{[\![B]\!]_{\langle \eta, a\rangle}} = R_{[\![B]\!]_{\langle \eta, b\rangle}}$, then*
$R_{[\![\Pi x{:}A.\,B]\!]_\eta} = \Pi(R_{[\![A]\!]_\eta}, a \mapsto R_{[\![B]\!]_{\langle \eta, a\rangle}})$.

SUBSET $R_{[\![\mathit{Power}(C)]\!]_\eta} = \mathcal{P}\mathrm{wr}(R_{[\![C]\!]_\eta})$.

ABS *If $\forall d, e.\ \ d\ R_{[\![A]\!]_\eta}\ e \Longrightarrow [\![M]\!]\langle \eta, d\rangle\ \ R_{[\![B]\!]_{\langle \eta, d\rangle}}\ \ [\![N]\!]\langle \eta, e\rangle$, then*
$\forall d, e.\ \ d\ R_{[\![A]\!]_\eta}\ e \Longrightarrow \mathit{App}([\![\lambda x{:}A.\,M]\!]_\eta, d)\ \ R_{[\![B]\!]_{\langle \eta, d\rangle}}\ \ [\![N]\!]\langle \eta, e\rangle$.

THIN *If Φ is a renaming on $\mathit{Dom}(\Gamma_1)$ such that $\Phi(\Gamma_1) \subseteq \Gamma_2$, then $[\![\Phi(\Gamma_1) \blacktriangleright \Phi(M) : \tau]\!]_{\eta_1} = [\![\Gamma_2 \blacktriangleright M : \tau]\!]_{\eta_2}$*

SUBST *If $\Gamma_1 \equiv \Gamma, x : A, \Gamma'$, $\Gamma_2 \equiv \Gamma, \Gamma'[N/x]$ then $[\![\Gamma_1 \blacktriangleright M]\!]_{\eta_1} = [\![\Gamma_2 \blacktriangleright M[N/x]]\!]_{\eta_2}$ provided ${\eta_1}^{\Gamma_1}(x) = [\![\Gamma \blacktriangleright N]\!]_{\eta_1|_x} \in \mathit{Rel}([\![\Gamma \blacktriangleright A]\!]_{\eta_1|_x})$.* □

Axioms CONST, VAR, APP are standard. $\mathsf{CONST2}$ requires that atomic types denote PERs. FAMILY and SUBSET define the extension of the denotation of types of the form $\Pi x{:}A.\,B$ and $\mathit{Power}(C)$. ABS ensures the soundness of the three equality rules which mention the λ-constructor.

Example 6.7 (Full hierarchy model). We define an interpretation by:

$$
\begin{array}{lcl}
\mathsf{Rel}(A) & = & A \\
[\![\Gamma \blacktriangleright x : \tau \;]\!]_\eta = & & \eta^\Gamma(x) \\
[\![\Gamma \blacktriangleright \kappa : \mathsf{P}(\kappa)]\!]_\eta & = & R_\kappa \\
[\![\Gamma \blacktriangleright \lambda x{:}A.\, M : \tau \Rightarrow \upsilon]\!]_\eta & = & a \mapsto [\![\Gamma, x : A \blacktriangleright M : \upsilon]\!]_{\langle \eta, a \rangle} \\
[\![\Gamma \blacktriangleright M\, N : \upsilon]\!]_\eta & = & \mathsf{App}([\![\Gamma \blacktriangleright M : \tau \Rightarrow \upsilon]\!]_\eta, [\![\Gamma \blacktriangleright N : \tau]\!]_\eta) \\
[\![\Gamma \blacktriangleright \Pi x{:}A.\, B : \mathsf{P}(\tau \Rightarrow \upsilon)]\!]_\eta & = & \Pi(R_{[\![A]\!]_\eta}, a \mapsto R_{[\![B]\!]_{\langle \eta, a \rangle}}) \\
[\![\Gamma \blacktriangleright \mathsf{Power}(A) : \mathsf{P}(\mathsf{P}(\tau))]\!]_\eta & = & \mathcal{P}wr(R_{[\![A]\!]_\eta}) \qquad \square
\end{array}
$$

Lemma 6.8. *The interpretation defined in Example 6.7 is a model of* λ_{Power}.

Here, $R_{[\![\Pi x:A.\, B]\!]_\eta}$ is not automatically a PER, since the uniformity condition that $a \;\; R_{[\![A]\!]_\eta} \;\; b \implies R_{[\![B]\!]_{\langle \eta, a \rangle}} = R_{[\![B]\!]_{\langle \eta, b \rangle}}$ may fail. For example, if $B \equiv zx$, the "rough-soundness" requirement that $\eta(z) \in \mathcal{D}^{\tau \Rightarrow \mathsf{P}(\upsilon)}$ does not force the value of z at one element of $\mathcal{D}^\tau$ to be related to the value at another. This is why we generalized to relations. PERs are only guaranteed for *well-formed* terms.

6.4 Soundness

Here we show that when $\Gamma \triangleright M : A$, then $[\![M]\!]_\eta$ is in the domain of the relation $R_{[\![A]\!]_\eta}$, and when $\Gamma \triangleright M = N : A$, then $[\![M]\!]_\eta$ is related to $[\![N]\!]_\eta$ by $R_{[\![A]\!]}$. Moreover, $\mathsf{Rel}([\![A]\!]_\eta)$ is a PER on $\mathcal{D}^\tau$, where $\Gamma \blacktriangleright A : \mathsf{P}(\tau)$. But we can only expect soundness if the environment η satisfies the context in a suitable way. The interpretation of a context Γ is defined by combining the interpretations of its components. Let S and T be sets, $R \in \mathrm{REL}(S)$, and $G \in dom(R) \to \mathrm{REL}(T)$. Then we define $\Sigma(R, G) \in \mathrm{REL}(S \times T)$ by:

$$p \;\; \Sigma(R,G) \;\; q \qquad \text{iff} \qquad \pi_1(p) \;\; R \;\; \pi_1(q) \quad \text{and} \quad \pi_2(p) \;\; G(\pi_1(p)) \;\; \pi_2(q)$$

Fact 6.9. *If* $R \in \mathrm{PER}(S)$ *and* $G(a) = G(b) \in \mathrm{PER}(T)$ *whenever* $a \;\; R \;\; b$, *then* $\Sigma(R, G) \in \mathrm{PER}(S \times T)$.

Definition 6.10 (Interpretation of contexts). *Given a model for* $\mathcal{D}$ *and a roughly-typable context* Γ, *we define* $[\![\Gamma]\!] \in \mathrm{REL}(\mathcal{D}^\Gamma)$ *by induction on* Γ, *by* $[\![\langle\rangle]\!] = \{\, (\star, \star) \,\}$ *and* $[\![\Gamma', x : A]\!] = \Sigma([\![\Gamma']\!], \eta \mapsto R_{[\![\Gamma' \blacktriangleright A : \mathsf{P}(\tau)]\!]_\eta})$. *We say* $\eta_1, \eta_2 \in \mathcal{D}^\Gamma$ *are* related environments satisfying Γ *iff* $\eta_1 \;\; [\![\Gamma]\!] \;\; \eta_2$.

Fact 6.11. *If* Γ *is roughly-typable and* $\eta_1 \;\; [\![\Gamma]\!] \;\; \eta_2$, *then for all* $x \in Dom(\Gamma)$, $\eta_1^\Gamma(x) \;\; R_{[\![\Gamma|_x \blacktriangleright \Gamma(x) : \mathsf{P}(\tau)]\!]_{\eta_1|_x}} \;\; \eta_2^\Gamma(x)$.

Lemma 6.12. *Suppose that* $R_{[\![A]\!]_\eta} \in \mathrm{PER}(\mathcal{D}^\tau)$ *and that* $d \;\; R_{[\![A]\!]_\eta} \;\; e \implies R_{[\![B]\!]_{\langle \eta, d \rangle}} = R_{[\![B]\!]_{\langle \eta, e \rangle}} \in \mathrm{PER}(\mathcal{D}^\upsilon)$ *for all* d, e. *Then in any model:*

WEAK-EXT *If* $\forall d, e.\ d\ R_{[\![A]\!]_\eta}\ e \Longrightarrow [\![M]\!]\langle \eta, d\rangle\ R_{[\![B]\!]\langle \eta,\, d\rangle}\ [\![N]\!]\langle \eta, e\rangle$, *then*
$\forall d, e.\ d\ R_{[\![A]\!]_\eta}\ e \Longrightarrow \mathsf{App}([\![\lambda x{:}A.\, M]\!]_\eta, d)\ R_{[\![B]\!]\langle \eta,\, d\rangle}\ \mathsf{App}([\![\lambda x{:}A.\, N]\!]_\eta, e)$

ETA *If* $\forall d, e.\ d\ R_{[\![A]\!]_\eta}\ e \Longrightarrow \mathsf{App}([\![M]\!]_\eta, d)\ R_{[\![B]\!]\langle \eta,\, d\rangle}\ \mathsf{App}([\![N]\!]_\eta, e)$, *then*
$\forall d, e.\ \ d\ R_{[\![A]\!]_\eta}\ e \Longrightarrow \mathsf{App}([\![\lambda x{:}A.\, M\, x]\!]_\eta, d)\ R_{[\![B]\!]\langle \eta,\, d\rangle}\ \mathsf{App}([\![N]\!]_\eta, e)$

Theorem 6.13 (Soundness for models).

1. *If* $\triangleright\ \Gamma$ *then* $[\![\Gamma]\!] \in \mathrm{PER}(\mathcal{D}^\Gamma)$.
2. *If* $\Gamma \triangleright M : A$, *then* $\forall \eta_1, \eta_2 \in \mathcal{D}^\Gamma$, $\eta_1\ [\![\Gamma]\!]\ \eta_2 \Longrightarrow [\![M]\!]_{\eta_1}\ R_{[\![A]\!]_{\eta_1}}\ [\![M]\!]_{\eta_2}$.
3. *If* $\Gamma \triangleright M = N : A$, *then* $\forall \eta_1, \eta_2 \in \mathcal{D}^\Gamma$, $\eta_1\ [\![\Gamma]\!]\ \eta_2 \Longrightarrow [\![M]\!]_{\eta_1}\ R_{[\![A]\!]_{\eta_1}}\ [\![N]\!]_{\eta_2}$.

Corollary 6.14 (Soundness of Typing). *If* $\Gamma \triangleright M : A$, *then for all* η, $\eta \in dom\,[\![\Gamma]\!] \implies [\![M]\!]_\eta \in R_{[\![A]\!]_\eta}$.

7 Conclusions

This paper introduces the type system λ_{Power}, a predicative fragment of Cardelli's original power type system [4]. Power types provide a cunning way of dealing with the subtyping judgement at the same time as the typing judgement. At first sight it appears to be a simplification, because two separate concerns are combined into one. However, the generalisation which occurs from using $\mathsf{Power}(A)$ as both a term and a type leads to complication of the meta-theory.

The semantics of λ_{Power} is set-based, but uses partial equivalence relations to interpret equality. The subtyping relation induced by power types is understood as inclusion between PERs. In contrast to other semantics for subtyping or dependent types, the intended model is made by "carving out" from a classical set-hierarchy, without using a universal domain. Every term in λ_{Power} has a *rough type* which is either an atomic type, or one of the forms $\tau \Rightarrow \upsilon$ or $\mathsf{P}(\tau)$, where τ and υ are rough types. These rough types are used to structure the set hierarchy.

Several important results are established. Unfortunately, there are still gaps in the meta-theory of λ_{Power}: ideally, we would like to prove a generation principle and thus prove subject reduction for λ_{Power}, which seems less straightforward than might be hoped (but no counterexamples have been found).

Acknowledgements

I'm especially grateful to Don Sannella who suggested that I study ASL+ [12], from which λ_{Power} grew. The work here first appeared in my thesis supervised by Sannella [1]. I am grateful to many people I discussed this work with during its evolution, including L. Cardelli, A. Compagnoni, H. Goguen, M. Hofmann, B. Pierce, and A. Tarklecki. More recently, the referees for CSL 2000 provided very useful and detailed comments (some questions are addressed only in [2] due to lack of space here).

$$\frac{\triangleright\ \Gamma}{\Gamma\ \triangleright\ \kappa : \mathsf{Power}(\kappa)}\ (\textsc{atomic})$$

$$\frac{\triangleright\ \Gamma \qquad x \in Dom(\Gamma)}{\Gamma\ \triangleright\ x : \Gamma(x)}\ (\textsc{var})$$

$$\frac{\Gamma,\, x : A\ \triangleright\ M : B}{\Gamma\ \triangleright\ \lambda x{:}A.\, M : \Pi x{:}A.\, B}\ (\lambda)$$

$$\frac{\Gamma\ \triangleright\ M : \Pi x{:}A.\, B \qquad \Gamma\ \triangleright\ N : A}{\Gamma\ \triangleright\ M\, N : B[N/x]}\ (\textsc{app})$$

$$\frac{\Gamma\ \triangleright\ M : A \qquad \Gamma\ \triangleright\ A : \mathsf{Power}(B)}{\Gamma\ \triangleright\ M : B}\ (\textsc{sub})$$

$$\frac{\Gamma\ \triangleright\ M : \Pi\vec{x}{:}\vec{A}.\ \mathsf{Power}(B)}{\Gamma\ \triangleright\ M : \Pi\vec{x}{:}\vec{A}.\ \mathsf{Power}(M\,\vec{x})}\ (\textsc{refl})$$

$$\frac{\Gamma\ \triangleright\ M : A \qquad \Gamma\ \triangleright\ A = B : \mathsf{Power}(C)}{\Gamma\ \triangleright\ M : B}\ (\textsc{conv})$$

$$\frac{\begin{array}{l}\Gamma\ \triangleright\ A' : \mathsf{Power}(A)\\ \Gamma,\, x : A'\ \triangleright\ B : \mathsf{Power}(B')\\ \Gamma,\, x : A\ \triangleright\ B : \mathsf{Power}(C)\end{array}}{\Gamma\ \triangleright\ \Pi x{:}A.\, B : \mathsf{Power}(\Pi x{:}A'.\, B')}\ (\Pi)$$

$$\frac{\Gamma\ \triangleright\ A : \mathsf{Power}(B)}{\Gamma\ \triangleright\ \mathsf{Power}(A) : \mathsf{Power}(\mathsf{Power}(B))}\ (\mathsf{Power})$$

Fig. 1. Typing rules

$$\frac{}{\triangleright\ \langle\rangle}\ (\textsc{empty})$$

$$\frac{\triangleright\ \Gamma \qquad \Gamma\ \triangleright\ A : \mathsf{Power}(B)}{\triangleright\ \Gamma,\, x : A}\ (\textsc{extend})$$

$$\frac{\Gamma\ \triangleright\ M : A}{\Gamma\ \triangleright\ M = M : A}\ (\textsc{eq-refl})$$

$$\frac{\Gamma\ \triangleright\ N = M : A}{\Gamma\ \triangleright\ M = N : A}\ (\textsc{eq-sym})$$

$$\frac{\Gamma\ \triangleright\ M = N : A \qquad \Gamma\ \triangleright\ N = P : A}{\Gamma\ \triangleright\ M = P : A}\ (\textsc{eq-trans})$$

$$\frac{\Gamma,\, x : A\ \triangleright\ M = M' : B}{\Gamma\ \triangleright\ \lambda x{:}A.\, M = \lambda x{:}A.\, M' : \Pi x{:}A.\, B}\ (\textsc{eq-}\lambda)$$

$$\frac{\begin{array}{l}\Gamma\ \triangleright\ M = M' : \Pi x{:}A.\, B\\ \Gamma\ \triangleright\ N = N' : A\end{array}}{\Gamma\ \triangleright\ M\, N = M'\, N' : B[N/x]}\ (\textsc{eq-app})$$

$$\frac{\Gamma,\, x : A\ \triangleright\ M : B \qquad \Gamma\ \triangleright\ N : A}{\Gamma\ \triangleright\ (\lambda x{:}A.\, M)\, N = M[N/x] : B[N/x]}\ (\textsc{eq-}\beta)$$

$$\frac{\Gamma\ \triangleright\ M : \Pi x{:}A.\, B}{\Gamma\ \triangleright\ \lambda x{:}A.\, M\, x = M : \Pi x{:}A.\, B}\ (\textsc{eq-}\eta)$$

Fig. 2. Context and equality rules

$$\frac{}{\blacktriangleright\ \langle\rangle}$$

$$\frac{\blacktriangleright\ \Gamma \qquad \Gamma\ \blacktriangleright\ A :\ P(\tau)}{\blacktriangleright\ \Gamma,\, x : A}$$

$$\frac{\blacktriangleright\ \Gamma}{\Gamma\ \blacktriangleright\ \kappa :\ P(\kappa)}$$

$$\frac{\blacktriangleright\ \Gamma \qquad \Gamma|_x\ \blacktriangleright\ \Gamma(x) :\ P(\tau)}{\Gamma\ \blacktriangleright\ x : \tau}$$

$$\frac{\Gamma\ \blacktriangleright\ A :\ P(\tau) \qquad \Gamma,\, x : A\ \blacktriangleright\ M : \upsilon}{\Gamma\ \blacktriangleright\ \lambda x{:}A.\, M : \tau \Rightarrow \upsilon}$$

$$\frac{\Gamma\ \blacktriangleright\ M : \tau \Rightarrow \upsilon \qquad \Gamma\ \blacktriangleright\ N : \tau}{\Gamma\ \blacktriangleright\ M\, N : \upsilon}$$

$$\frac{\Gamma\ \blacktriangleright\ A :\ P(\tau) \qquad \Gamma,\, x : A\ \blacktriangleright\ B :\ P(\upsilon)}{\Gamma\ \blacktriangleright\ \Pi x{:}A.\, B :\ P(\tau \Rightarrow \upsilon)}$$

$$\frac{\Gamma\ \blacktriangleright\ A :\ P(\tau)}{\Gamma\ \blacktriangleright\ \mathsf{Power}(A) :\ P(P(\tau))}$$

Fig. 3. Rough typing rules

References

[1] David Aspinall. *Type Systems for Modular Programs and Specification.* PhD thesis, Department of Computer Science, University of Edinburgh, 1997.

[2] David Aspinall. Subtyping with power types. Draft full version. LFCS, University of Edinburgh, 2000.

[3] David Aspinall and Adriana Compagnoni. Subtyping dependent types. In E. Clarke, editor, *Proceedings, Eleventh Annual IEEE Symposium on Logic in Computer Science*, pages 86–97, New Brunswick, New Jersey, 1996. IEEE Computer Society Press.

[4] Luca Cardelli. Structural subtyping and the notion of power type. In *Conference Record of the Fifteenth Annual ACM Symposium on Principles of Programming Languages*, pages 70–79, San Diego, California, January 13–15, 1988. ACM SIGACT-SIGPLAN, ACM Press.

[5] Luca Cardelli. Notes about $F^{\omega}_{<:}$. Unpublished manuscript, October 1990.

[6] Luca Cardelli and Giuseppe Longo. A semantic basis for Quest. *Journal of Functional Programming*, 1(4):417–458, 1991.

[7] Karl Crary. *Type-theoretic Methodology for Practical Programming Languages.* PhD thesis, Cornell University, 1998.

[8] Philippa Gardner. *Representing Logics in Type Theory.* PhD thesis, Department of Computer Science, University of Edinburgh, 1992.

[9] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *JACM*, 40(1):143–184, 1993.

[10] John C. Mitchell. *Foundations for Programming Languages.* MIT Press, 1996.

[11] Frank Pfenning. Refinement types for logical frameworks. In *Informal Proceedings of the 1993 Workshop on Types for Proofs and Programs*, pages 315–328, May 1993.

[12] Donald Sannella, Stefan Sokołowski, and Andrzej Tarlecki. Toward formal development of programs from algebraic specifications: Parameterisation revisited. *Acta Informatica*, 29:689–736, 1992.

The Descriptive Complexity of the Fixed-Points of Bounded Formulas

Albert Atserias*

Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
c/ Jordi Girona Salgado, 1-3, Edif. C6.
08034 Barcelona, Spain.
atserias@lsi.upc.es

Abstract. We investigate the complexity of the fixed-points of bounded formulas in the context of finite set theory; that is, in the context of arbitrary classes of finite structures that are equipped with a built-in BIT predicate, or equivalently, with a built-in membership relation between hereditarily finite sets (input relations are allowed). We show that the iteration of a positive bounded formula converges in polylogarithmically many steps in the cardinality of the structure. This extends a previously known much weaker result. We obtain a number of connections with the rudimentary languages and deterministic polynomial-time. Moreover, our results provide a natural characterization of the complexity class consisting of all languages computable by bounded-depth, polynomial-size circuits, and polylogarithmic-time uniformity. As a byproduct, we see that this class coincides with LH(P), the logarithmic-time hierarchy with an oracle to deterministic polynomial-time. Finally, we discuss the connection of this result with the well-studied algorithms for integer division.

Keywords: Circuit uniformity, BIT predicate, logarithmic-time hierarchy, rudimentary languages, integer division.

1 Introduction

1.1 Background

The Ordered Conjecture of Kolaitis and Vardi [22] states that least fixed-point logic LFP is strictly more expressive than first-order logic FO on every infinite class of ordered finite structures. Informally, the conjecture expresses an inherent limitation of first-order logic to capture polynomial-time computations on finite structures, no-matter how rich the combinatorial nature of the structures is. The question remains open, and it is known that any way of solving it will have important consequences in Complexity Theory. A refutation would imply that **P** $\neq$ **PSPACE** [12], and a proof would imply that **LINH** $\neq$ **E** [13]. Here, **LINH**

* Supported by the CUR, Generalitat de Catalunya, through grant 1999FI 00532, and partially supported by ALCOM-FT, IST-99-14186.

P. Clote and H. Schwichtenberg (Eds.): CSL 2000, LNCS 1862, pp. 172–186, 2000.

is the linear-time hierarchy of Wrathall [35], and **E** is the usual complexity class that consists of all languages that are accepted by deterministic Turing machines in time $2^{O(n)}$.

There is a special case of the conjecture, singled out by Gurevich, Immerman, and Shelah [15], that is of particular interest. Namely, it is unknown whether LFP collapses to FO on the class of all finite structures of the form $(\{0,\ldots,n-1\},\leq,\mathrm{BIT})$, where $\leq$ is the usual linear ordering, and BIT is the binary relation that consists of all pairs (p,q) of natural numbers such that the p-th bit in the binary expansion of q is one. As pointed out in [15], the collapse happens if and only if **DLOGTIME**-uniform $\mathbf{AC}^0 = \mathbf{P}$-uniform $\mathbf{AC}^0$ (see Section 2 for definitions), or equivalently, if and only if **LINH** = **E**. Motivated by this interesting connection, Atserias and Kolaitis [3] investigated the difficulty of settling this special case of the Ordered Conjecture. Their approach is further motivated by the existence of a well-known isomorphism between $(\mathbb{N},\mathrm{BIT})$ and $(V_\omega,\in)$ (see [6]), where V_ω is the class of all hereditarily finite sets; that is, $V_\omega = \bigcup_{n\geq 0} V_n$, where $V_{n+1} = \mathcal{P}(V_n)$ and $V_0 = \emptyset$. The Ackermann bijection $e : \mathbb{N} \to V_\omega$ defined for every $n \in \mathbb{N}$ as

$$e(n) = \{e(m) : \text{the } m\text{-th bit of } n \text{ is one}\},$$

is the aforementioned isomorphism. Furthermore, by exploiting this mapping of BIT into $\in$, Dawar, Doets, Lindell and Weinstein [11] showed the somewhat surprising result that the standard linear order is first-order definable from the BIT predicate alone. Hence, the question translates into whether LFP collapses to FO on the class $\mathcal{BFR} = \{(\{e(0),\ldots,e(n-1)\},\in) : n \geq 0\}$. In view of the Ackermann bijection, we identify the structures $(\{0,\ldots,n-1\},\mathrm{BIT})$ and $(\{e(0),\ldots,e(n-1)\},\in)$, here and in the future, and thus use the notation $\mathbf{BIT}_n = (\{0,\ldots,n-1\},\in)$.

This *set-theoretic* framework led to the study of the fixed-points of the Δ_0 formulas of set theory, also called bounded formulas. These are the formulas all of whose quantifiers are of the form $(\exists x \in y)$ and $(\forall x \in y)$ (Sazonov has studied the fixed-points of bounded formulas in the context of definability on $(V_\omega,\in)$, rather than in the context of uniform definability on finite structures; see [30] for a survey). It was proved by Atserias and Kolaitis [3] that if the fixed-points of positive Δ_0 formulas $\mathrm{LFP}(\Delta_0)$ were first-order definable on $\mathcal{BFR}$, then $\mathbf{P} \subseteq \mathbf{LINH}$ and so $\mathbf{P} \neq \mathbf{PSPACE}$. Thus, settling whether $\mathrm{LFP}(\Delta_0)$ collapses is already a difficulty question. Nonetheless, the authors were able to show that the fixed-points of the so-called *restricted* Δ_0 formulas were indeed first-order definable, and so were the fixed-points of all unary and binary Δ_0 formulas. Finally back to complexity issues, they showed that the number of times that a positive Δ_0 formula has to be iterated until its fixed-point is reached in a finite structure, its closure function, is bounded by a polylogarithm of the cardinality of the structure on a small subclass of $\mathcal{BFR}$. As a consequence, these fixed-points are computable in **NC** on this class.

1.2 Main Results

The isomorphism mapping BIT to the membership relation $\in$ constitutes a good source of inspiration to obtain results that explain the expressive power of first-order logic and fixed-point logic when strong built-in relations are available. The results in [11] and [3] are good examples. Moreover, the set-theoretic framework provides new concepts to consider, such as Δ_0 formulas, and new techniques to apply, such as absoluteness arguments. However, the complexity aspects of $\mathrm{LFP}(\Delta_0)$ were not completely studied in [3], and we feel that the results of the present paper complete this study.

The first result of this paper is the extension of the last result in [3] to arbitrary classes of finite structures with built-in membership (BIT) relation. That is, we show that the closure functions of Δ_0 formulas are bounded by a polylogarithm of the cardinality of the universe of any arbitrary finite structure with built-in membership relation. Moreover, we observe that this implies that $\mathrm{LFP}(\Delta_0)$ is computable in **DPOLYLOGTIME** (and not simply in **NC**) on any arbitrary class of finite structures with built-in membership relation. Then we focus back to the class $\mathcal{BFR}$. We observe that on this particular class, $\mathrm{LFP}(\Delta_0)$ is even in (non-uniform) $\mathbf{AC}^0$ for some trivial reasons. The interesting question is then: Which uniform version of $\mathbf{AC}^0$ is captured by $\mathrm{FO} + \mathrm{LFP}(\Delta_0)$, the first-order closure of $\mathrm{LFP}(\Delta_0)$? Our second main result is the answer to this question: on $\mathcal{BFR}$, the logic $\mathrm{FO} + \mathrm{LFP}(\Delta_0)$ captures **DPOLYLOGTIME**-uniform $\mathbf{AC}^0$ which in turn, coincides with $\mathbf{LH}^{\mathbf{P}}$; the logarithmic-time hierarchy of Sipser with an oracle to **P**. As a corollary we obtain an exact characterization of the complexity-theoretic difficulties of showing $\mathrm{FO} + \mathrm{LFP}(\Delta_0) = \mathrm{FO}$ on $\mathcal{BFR}$. We show that the collapse is equivalent to $\mathbf{P} \subseteq \mathbf{LINH}$. Note that **LINH** coincides with the rudimentary languages **RUD** [35] introduced by Smullyan [32].

We then consider the descriptive complexity of $\mathrm{FO} + \mathrm{LFP}(\Delta_0)$ on arbitrary classes of finite structures with built-in membership relation. Somewhat surprisingly, we are only able to provide an exact answer in the case that the underlying vocabulary of the class of structures is unary (on classes of words with built-in membership relation). In that case, $\mathrm{FO} + \mathrm{LFP}(\Delta_0)$ still captures **DPOLYLOGTIME**-uniform $\mathbf{AC}^0$. For higher arities, however, we are only able to compare the relative expressive power of $\mathrm{FO} + \mathrm{LFP}(\Delta_0)$ and FO with a complexity-theoretic question. We show that if $\mathbf{P} \subseteq \mathbf{RUD}_{n^{1/r}}$, then $\mathrm{FO} + \mathrm{LFP}(\Delta_0)$ collapses to FO on any arbitrary class of finite structures with built-in membership relation over a vocabulary of arity at most r. The class $\mathbf{RUD}_{n^{1/r}}$ was introduced by Jones [21] as a natural subclass of the rudimentary languages $\mathbf{RUD} = \mathbf{RUD}_n$. A result of Allender and Gore [1] implies that $\mathbf{RUD}_{n^\epsilon}$ coincides with $\mathrm{ATIME}(O(n^\epsilon), O(1))$ for every $\epsilon \in (0, 1]$. Here, $\mathrm{ATIME}(t(n), a(n))$ is the class of languages accepted by alternating Turing machines in time $t(n)$ and $a(n)$ alternations. Moreover, as mentioned by Allender and Gore, $\mathbf{RUD}_{n^\epsilon}$ contains complete problems of each level of the polynomial-time hierarchy **PH** [33, 35].

It is interesting that **DPOLYLOGTIME**-uniform $\mathbf{AC}^0$ comes out of our results as a natural complexity class (we note that polylogtime uniformity has

been considered at least once in the past by Allender and Gore [2], although in a completely different context). The reason amounts to a connection with the problem of the uniformity of Boolean circuits for integer division, an interesting issue that has received a good deal of attention [7, 28, 23, 19]. See the end of Section 5 for more details. Finally, it is obvious that our objects of study are intimately related to questions about the rudimentary languages, a well-studied topic [32, 8, 20, 35, 27, 1]. We point out that the rudimentary languages, and the techniques related to them, have been revisited very recently by Fortnow [14], and Lipton and Viglas [24], to obtain significant progress in some important open problems in Complexity Theory.

2 Preliminaries

Logic. Let $\sigma = \{R_1, \ldots, R_s\}$ be a finite relational vocabulary, and let $\mathbf{M} = (M, R_1^{\mathbf{M}}, \ldots, R_s^{\mathbf{M}})$ be a finite structure over σ. We will always identify the universe of $\mathbf{M}$, denoted M, with the initial segment of the natural numbers of cardinality $|M|$; thus, $M = \{0, \ldots, |M|-1\}$. Let $R = (R_1, R_2, \ldots)$ be a sequence of k-ary relations such that $R_n \subseteq \{0, \ldots, n-1\}^k$. Let C be a class of finite structures for $\sigma \cup \{R\}$, with $R \notin \sigma$. We say that C is a class of finite structures over σ with built-in R-relation if and only if, for every $\mathbf{M} \in C$, we have that $R^{\mathbf{M}} = R_{|M|}$. Notice that the built-in relation only depends on the cardinality of the structure.

Least fixed-point logic FO + LFP is the extension of first-order logic FO obtained by augmenting the syntax with a new formula $\mathrm{LFP}_{\overline{x},X}\varphi(x_1, \ldots, x_k, X)$, for every first-order formula φ positive in the k-ary relation variable X. The meaning of $\mathbf{M} \models (\mathrm{LFP}_{\overline{x},X}\varphi)[\overline{a}]$ is that $\overline{a} \in I_\varphi(\mathbf{M})$, where $I_\varphi(\mathbf{M})$ is the least fixed-point of the monotone operator defined by φ on $\mathbf{M}$. We let $I_\varphi^m(\mathbf{M})$ be the m-th stage, that is, $I_\varphi^m(\mathbf{M}) = \{\overline{a} \in M^k : \mathbf{M} \models \varphi[\overline{a}, \bigcup_{m'<m} I_\varphi^{m'}(\mathbf{M})]\}$. It is known that FO + LFP is closed under nested applications of the least fixed-point operator (see [17, 16]).

We let $\mathrm{LFP}(\Delta_0)$ be the class of formulas of the form $\mathrm{LFP}_{\overline{x},X}\varphi(x_1, \ldots, x_k, X)$, where φ is a Δ_0 formula positive in k-ary relation variable X. Observe that first-order parameters are not allowed, and neither is the nesting of fixed-point operators. We let $\mathrm{FO} + \mathrm{LFP}(\Delta_0)$ denote the closure of $\mathrm{LFP}(\Delta_0)$ under all first-order connectives and quantification.

Complexity. For every natural number n, we let $\log n$ denote the length of the shortest binary representation of n. If we wish to use the true base-two logarithm, we use the notation $\log_2(n)$; thus, $\log n = \lfloor \log_2(n) \rfloor + 1$. We identify natural numbers with their shortest binary representation. However, for every $m \in \{0, \ldots, n-1\}$, we let $\mathrm{b}_n(m)$ denote the unique binary representation of length $\log(n-1)$ (padded with leading zeros if necessary).

Our model of computation is the oracle alternating multitape Turing machine with random access to the input. This model, originally defined by Ruzzo [29] and used by Barrington, Immerman and Straubing [5], Buss [9], and Sipser

[31] among others, is a modification of the model of Chandra, Kozen and Stockmeyer [10] to allow sublinear time-bounds. These machines are equipped with an address tape on which to write a number in binary. When the machine enters a distinguished state with a number p written on its address tape, the head of the input tape jumps, in one step, to the p-th leftmost cell of the tape. Strictly speaking, the definition of Ruzzo [29] is slightly different from ours, but standard simulation arguments show that both models have the same computing power with only a constant factor loss in time or number of alternations (see [5] and [9] for example). In the case of deterministic machines, our model is slightly more robust, but this will not affect the generality of the results.

Finite structures are encoded as words over the alphabet $\{0, 1, \#\}$ according to the following convention. For every relation symbol $R_i \in \sigma$ of arity r, we let $\chi(R_i^{\mathbf{M}})$ be the characteristic sequence of $R_i^{\mathbf{M}}$. That is, $\chi(R_i^{\mathbf{M}}) = a_0 a_1 \dots a_{n^r-1}$, where $a_m \in \{0,1\}$, and $a_m = 1$ if and only if $(m_{r-1}, \dots, m_0) \in R_i^{\mathbf{M}}$ where $(m_{r-1}, \dots, m_0)$ is the n-ary representation of m. Then, the encoding of $\mathbf{M}$ is just

$$\langle \mathbf{M} \rangle = 1^n \# \chi(R_1^{\mathbf{M}}) \# \dots \# \chi(R_s^{\mathbf{M}}).$$

We extend the encoding to include individuals as follows. For every $a_1, \dots, a_k \in M$, let $\langle \mathbf{M}, a_1, \dots, a_k \rangle = \langle \mathbf{M} \rangle \# \mathrm{b}_n(a_1) \# \dots \# \mathrm{b}_n(a_k)$. Let C be a class of finite structures, and let Q be a k-ary query on C. We say that Q is computable in a complexity class $\mathcal{C}$ on C if there exists a language $L \in \mathcal{C}$ such that for every $\mathbf{M} \in C$ and $a_1, \dots, a_k \in M$, we have that $(a_1, \dots, a_k) \in Q(\mathbf{M})$ if and only if $\langle \mathbf{M}, a_1, \dots, a_k \rangle \in L$. We say that a k-ary built-in relation $R = (R_1, R_2, \dots)$ is computable in a complexity class $\mathcal{C}$ if there exists a language $L \in \mathcal{C}$ such that for every n and $a_1, \dots, a_k \in \{0, \dots, n-1\}$, we have that $(a_1, \dots, a_k) \in R_n$ if and only if $1^n \# \mathrm{b}_n(a_1) \# \dots \# \mathrm{b}_n(a_k) \in L$. When considering Boolean circuits, we are forced to restrict ourselves to the binary alphabet $\{0, 1\}$. We fix then an homomorphism $h : \{0, 1, \#\}^* \to \{0, 1\}^*$ in a standard way: put $h(0) = 00$, $h(1) = 11$ and $h(\#) = 01$ (see Section 4 for more details).

3 General Facts about Bounded Formulas

Recall that the transitive closure of a set a, denoted by $\mathrm{TC}(a)$, is defined inductively as follows: $\mathrm{TC}(a) = \bigcup\{\mathrm{TC}(b) : b \in a\}$. The reflexive transitive closure of a, denoted by $\mathrm{RTC}(a)$, is $\{a\} \cup \mathrm{TC}(a)$. Our first Lemma says that the satisfiability of a Δ_0 formula only depends on the reflexive transitive closure of its arguments. Given a first-order formula $\varphi(x_1, \dots, x_n)$ with free variables among $x_1, \dots, x_n$, we let $F(\varphi)$ be the set of *indices* of the free variables of φ.

Lemma 1. *Let σ be a relational vocabulary, let $\mathbf{M}$ be a structure for $\sigma \cup \{\in\}$ with built-in membership relation, and let $\varphi(x_1, \dots, x_s, X)$ be a Δ_0 formula over $\sigma \cup \{\in, X\}$, where X is a k-ary relation variable. For every $A \subseteq M^k$, and every tuple $\overline{a} = (a_1, \dots, a_s) \in M^s$, we have that $\mathbf{M} \models \varphi[\overline{a}, A]$ if and only if $\mathbf{M} \models \varphi[\overline{a}, A \cap (\bigcup\{\mathrm{RTC}(a_i) : i \in F(\varphi)\})^k]$.*

Proof: We proceed by induction on the construction of φ. The base cases are trivial, and so is the case in which φ is of the form $\neg\psi$. Suppose that φ is of the form $\psi_1 \wedge \psi_2$. Let $B = \bigcup\{\mathrm{RTC}(a_i) : i \in F(\varphi)\}$ and $B_j = \bigcup\{\mathrm{RTC}(a_i) : i \in F(\psi_j)\}$ for $j = 1, 2$. Then, $\mathbf{M} \models \varphi[\overline{a}, A \cap B^k]$ if and only if $\mathbf{M} \models \psi_j[\overline{a}, A \cap B^k]$ for $j = 1, 2$. By induction hypothesis, this is equivalent to $\mathbf{M} \models \psi_j[\overline{a}, A \cap B^k \cap B_j^k]$ for $j = 1, 2$. Since $F(\psi_j) \subseteq F(\varphi)$, this is equivalent to $\mathbf{M} \models \psi_j[\overline{a}, A \cap B_j^k]$ for $j = 1, 2$. By induction hypothesis again, this is equivalent to $\mathbf{M} \models \psi_j[\overline{a}, A]$ for $j = 1, 2$, and therefore to $\mathbf{M} \models \varphi[\overline{a}, A]$. Suppose next that φ is of the form $(\exists x_i \in x_j)\psi$. Let $B = \bigcup\{\mathrm{RTC}(a_i) : i \in F(\varphi)\}$. In this case, $\mathbf{M} \models \varphi[\overline{a}, A \cap B^k]$ if and only if there is some $a \in M$ such that $a \in a_j$ and $\mathbf{M} \models \psi[\overline{b}, A \cap B^k]$, where $\overline{b} = (a_1, \ldots, a_{i-1}, a, a_{i+1}, \ldots, a_s)$. Let $B(\overline{b}) = \bigcup\{\mathrm{RTC}(b_l) : l \in F(\psi)\}$. Therefore, by induction hypothesis, $\mathbf{M} \models \varphi[\overline{a}, A \cap B^k]$ if and only if there is some $a \in M$ such that $a \in a_j$ and $\mathbf{M} \models \psi[\overline{b}, A \cap B^k \cap B(\overline{b})^k]$. Since for every $a \in a_j$ we have that $\mathrm{RTC}(a) \subseteq \mathrm{RTC}(a_j)$, it is the case that $B(\overline{b}) \subseteq B$. Consequently, $\mathbf{M} \models \varphi[\overline{a}, A \cap B^k]$ if and only if there is some $a \in M$ such that $a \in a_j$ and $\mathbf{M} \models \psi[\overline{b}, A \cap B(\overline{b})^k]$, and by induction hypothesis again, $\mathbf{M} \models \psi[\overline{b}, A]$, as required. □

For every first-order formula $\varphi(x_1, \ldots, x_k, X)$ positive in the k-ary relation symbol X, we let $\mathrm{cl}_\varphi(\mathbf{M})$ denote the closure ordinal of φ in $\mathbf{M}$; that is, $\mathrm{cl}_\varphi(\mathbf{M})$ is the minimum ordinal α such that $I_\varphi^\alpha(\mathbf{M}) = \bigcup_{\alpha' < \alpha} \mathbf{I}_\varphi^{\alpha'}(\mathbf{M})$ [26]. Since the reflexive transitive closure of a finite set is relatively small, Lemma 1 allows us to put polylogarithmic bounds on the closure functions of Δ_0 formulas. This result extends Theorem 4 in [3] to the case of $\mathcal{BFR}$, and in fact, to arbitrary classes of finite structures with built-in membership relation.

Theorem 1. *Let σ be a relational vocabulary, and let $\varphi(x_1, \ldots, x_k, X)$ be a Δ_0 formula over $\sigma \cup \{\in, X\}$ that is positive in the k-ary relation variable X. Then,*

$$\mathrm{cl}_\varphi(\mathbf{M}) \leq (\log(|M| - 1) + k)^k$$

for every finite structure $\mathbf{M}$ over $\sigma \cup \{\in\}$ with built-in membership relation.

Proof: Put $t = (\log(|M|-1)+k)^k$, and assume for contradiction that $\mathrm{cl}_\varphi(\mathbf{M}) > t$. Let $\overline{a}_0 = (a_{0,1}, \ldots, a_{0,k}) \in I_\varphi(\mathbf{M})$ be such that $|\overline{a}_0| > t$, where $|\overline{a}|$ denotes the minimal m such that $\overline{a} \in I_\varphi^m(\mathbf{M})$ if $\overline{a} \in I_\varphi(\mathbf{M})$, and ∞ if $\overline{a} \notin I_\varphi(\mathbf{M})$. In the following, let I^m be an abbreviation for $I_\varphi^m(\mathbf{M})$. We build a sequence $\overline{a}_0, \overline{a}_1, \ldots, \overline{a}_t$ such that $|\overline{a}_i| = |\overline{a}_0| - i$, and $\overline{a}_i \in S^k$ for every $i = 0, \ldots, t$, where $S = \{0, \ldots, \log(|M|-1)-1\} \cup \{a_{0,1}, \ldots, a_{0,k}\}$. This will prove the theorem since the cardinality of S^k is at most t.

For every $\overline{a} = (a_1, \ldots, a_k) \in M^k$, let $S(\overline{a})$ denote the set $\{0, \ldots, \log(|M| - 1)\} \cup \{a_1, \ldots, a_k\}$. Observe that $\bigcup\{\mathrm{RTC}(a_i) : i \in F(\varphi)\} \subseteq S(\overline{a})$ since every element in $\mathrm{TC}(a_i)$ is a *bit* position of an element in $\{0, \ldots, |M| - 1\}$. Assuming $\overline{a}_i = (a_{i,1}, \ldots, a_{i,k})$ is already defined, we define $\overline{a}_{i+1} = (a_{i+1,1}, \ldots, a_{i+1,k})$. Let $m = |\overline{a}_i|$. Then, $\mathbf{M} \models \varphi[\overline{a}_i, I^{m-1}]$. Lemma 1 and monotonicity imply that $\mathbf{M} \models \varphi[\overline{a}_i, I^{m-1} \cap S(\overline{a}_i)^k]$. Observe that since $\overline{a}_i \in S(\overline{a}_0)^k$ by assumption, we have that $S(\overline{a}_i) \subseteq S(\overline{a}_0)$. Now let us consider two cases: (i) $I^{m-1} \cap S(\overline{a}_0)^k \subseteq I^{m-2}$, or (ii) $I^{m-1} \cap S(\overline{a}_0)^k \not\subseteq I^{m-2}$. In case (i) we have that $\mathbf{M} \models \varphi[\overline{a}_i, I^{m-2}]$ by

monotonicity. Hence, $\overline{a}_i \in I^{m-1}$ which contradicts the minimality of $m = |\overline{a}_i|$. In case (ii), there must exist some $\overline{a}_{i+1} \in I^{m-1} \cap S(\overline{a}_0)^k$ that does not belong to I^{m-2}. Observe that $|\overline{a}_{i+1}| = |\overline{a}_i| - 1$, and $\overline{a}_{i+1} \in S(\overline{a}_0)^k$ as required. This completes the proof of the theorem. □

The polylogarithmic bounds on the closure functions, together with a result of Immerman [18], imply that every query that is definable as the fixed-point of a Δ_0 formula is computable in **NC**, the parallel complexity class. However, we can keep the machine sequential as noted in the following

Lemma 2. *Let σ be a relational vocabulary, let C be a class of finite structures over σ with built-in membership relation, and let $\varphi(x_1, \ldots, x_k, X)$ be a Δ_0 formula that is positive in the k-ary relation variable X. Then, the query on C defined by the formula $(\mathrm{LFP}_{\overline{x},X}\varphi)$ is computable in* **DPOLYLOGTIME** *on C.*

Proof: The idea is that the standard fixed-point computation will only take a polylogarithmic number of iterations by Theorem 1, and each iteration is computable in polylogarithmic-time because φ is a Δ_0 formula. More precisely, on input $\langle \mathbf{M}, a_1, \ldots, a_k \rangle$, the polylogarithmic-time Turing machine will proceed as follows. The machine first determines the cardinality of M, say n. To this end, it determines the length m of the input in $O(\log m)$ steps using its random access to the input (see [5] for this trick), and then it executes a straightforward computation to extract n from m (here we use the fact that our encodings are carefully chosen so that their length is determined by the cardinality of $\mathbf{M}$, the signature of σ, and k). Let $B = \{0, \ldots, \log(n-1) - 1\} \cup \{a_1, \ldots, a_k\}$. The machine will keep, in a separate tape, an encoding of a k-ary relation on B; this will require $O((\log n)^k)$ bits of information. Then, it starts a loop that is to be repeated $(\log(n-1) + k)^k$ times. In each iteration, the machine cycles through all k-tuples $(b_1, \ldots, b_k)$ in B^k, and evaluates $\mathbf{M} \models \varphi[b_1, \ldots, b_k, R]$ where R is the k-ary relation encoded in the separate tape. Atomic formulas from σ are resolved by random access to the input, and atomic formulas of the form $X(u_1, \ldots, u_k)$ are resolved by accessing the position of tuple $(u_1, \ldots, u_k)$ in the encoding of R. Observe that each relevant tuple $(u_1, \ldots, u_k)$ will be available since $\mathbf{M} \models \varphi[b_1, \ldots, b_k, R]$ if and only if

$$\mathbf{M} \models \varphi[b_1, \ldots, b_k, R \cap (\bigcup\{\mathrm{RTC}(b_i) : i \in F(\varphi)\})^k],$$

and $\bigcup\{\mathrm{RTC}(b_i) : i \in F(\varphi)\} \subseteq B$ for every $(b_1, \ldots, b_k) \in B^k$, since every element of $\mathrm{TC}(a_i)$ is a *bit* position of an element in $\{0, \ldots, n-1\}$. For the same reason, each quantifier is bounded by some b_i, and therefore, the variable it bounds ranges over at most $\log(n-1)$ elements of the universe. Hence, the computation can be done in time $O((\log n)^r)$ where r depends on the number of quantifiers of φ. When the evaluation of $\mathbf{M} \models \varphi[b_1, \ldots, b_k, R]$ is complete, the machine updates accordingly the position corresponding to tuple $(b_1, \ldots, b_k)$ in the encoding of R. Finally, the machine will only have to check whether the tuple $(a_1, \ldots, a_k)$ belongs to R at the end of the loop. □

4 Fixed-Points of Bounded Formulas on $\mathcal{BFR}$

A closer examination of Lemma 2 in the case of $\mathcal{BFR}$ reveals that LFP(Δ_0)-definable queries are also computable in (non-uniform) $\mathbf{AC}^0$; the reason is that they only depend on $O(\log n)$ bits of the input (in fact, the relevant part of the input is that short already). The interesting question at this point is the following: which uniform version of $\mathbf{AC}^0$ is captured by FO+LFP(Δ_0) on $\mathcal{BFR}$? Our next theorem is the answer to this question. Before stating the result, we need some definitions.

Let $C = (C_1, C_2, \ldots)$ be a sequence of boolean circuits, and let s_n be a bound on the size of C_n. Thus, gates in C_n may be numbered in $\{0, \ldots, s_n - 1\}$. The direct connection language of C (see [5]) is the set of words of the form $1^n \# \mathrm{b}_{s_n}(a) \# \mathrm{b}_{s_n}(b) \# t$, where gate b is an input to gate a, and the type of gate b is $t \in \{0, 1, 2, 3\}$. Here, $t = 0$ means that b is an AND gate, $t = 1$ means that b is an OR gate, $t = 2$ means that b is a positive input, and $t = 3$ means that b is a negated input. If $\mathcal{C}$ is a complexity class, we say that C is $\mathcal{C}$-uniform if there exists a language in $L \in \mathcal{C}$ such that for every word w of the form $1^n \# a \# b \# t$ we have that $w \in L$ if and only if $w \in \mathrm{DCL}(C)$. The class $\mathcal{C}$-uniform $\mathbf{AC}^0$ is the class of all languages that are accepted by a $\mathcal{C}$-uniform, polynomial-size, bounded-depth, family of circuits (for languages $L \subseteq \Sigma^*$ with $\Sigma \neq \{0, 1\}$, we say that L is accepted by a family of circuits C if $h(L)$ is accepted by C for some fixed homomorphism $h : \Sigma^* \to \{0, 1\}^*$ [5]).

Theorem 2. *Let Q be a query on $\mathcal{BFR}$. The following are equivalent:*

1. *Q is computable in $\mathbf{LH}^{\mathbf{P}}$ on $\mathcal{BFR}$,*
2. *Q is computable in* **DPOLYLOGTIME***-uniform $\mathbf{AC}^0$ on $\mathcal{BFR}$,*
3. *Q is definable in $\mathrm{FO} + \mathrm{LFP}(\Delta_0)$ on $\mathcal{BFR}$.*

Proof: We close a cycle of implications. We first show that (i) implies (ii). Assume that Q is computable in $\mathbf{LH}^A$ for some $A \in \mathbf{P}$. We may assume $A \subseteq \{0, 1\}^*$. For every n, let $F_n(x_1, \ldots, x_n)$ be the following DNF-formula

$$\bigvee_{\overline{a} \in A \cap \{0,1\}^n} \left(\bigwedge_{a_i = 1} x_i \wedge \bigwedge_{a_i = 0} \neg x_i \right).$$

Observe that $F_n(a_1, \ldots, a_n)$ is true if and only if the word $a_1 \ldots a_n$ belongs to A. The sequence $(F_1, F_2, \ldots)$, interpreted as a sequence of depth-two circuits, is exponential-size in n, but $\mathbf{P}$-uniform (the words of its direct connection language are of the form $1^n \# a \# b \# t$ with $a, b \in \{0, 1\}^{O(n)}$, and deciding membership can be done in polynomial-time since $A \in \mathbf{P}$). We now build a **DPOLYLOGTIME**-uniform family of $\mathbf{AC}^0$ circuits to compute Q. Let M be an oracle alternating Turing machine witnessing that Q is computable in $\mathbf{LH}^A$, and assume that M queries its oracle at most once in each computation path (this is a standard trick in alternating machines; it consists of existentially guessing the answers, write them down on a separate tape together with the nondeterministic branch taken at each step, and at the end of the computation, universally branch to

check the correctness of every guess by deterministically resimulating the computation path until the challenged query is asked). Let $c \log n$ be a bound on the running-time of M on inputs of length n. Observe that the length of each oracle query is bounded by $c \log n$ too. As in [5], we may see the computation trees of M as a **DLOGTIME**-uniform family of $\mathbf{AC}^0$ circuits, except for the oracle queries, which may be resolved by **DPOLYLOGTIME**-uniform $\mathbf{AC}^0$ circuits; namely, we let queries of length $m \leq c \log n$ be resolved by the circuit F_m above (exponential-size in $m \leq c \log n$ is polynomial-size in n, and polynomial-time uniformity for length $m \leq c \log n$ is polylogarithmic-time uniformity for length n). It follows that Q is computable in **DPOLYLOGTIME**-uniform $\mathbf{AC}^0$.

We see that (ii) implies (iii). Let Q be computable in **DPOLYLOGTIME**-uniform $\mathbf{AC}^0$ (recall the convention established just before the statement of the theorem). It is well-known [4,25] that Q is then first-order definable with an additional built-in relation $R = (R_1, R_2, \ldots)$ that is computable in polylogarithmic-time. We show how to replace every occurrence of this built-in relation by a formula of $\mathrm{FO} + \mathrm{LFP}(\Delta_0)$. For every n and $\overline{a} = (a_1, \ldots, a_k) \in \{0, \ldots, n-1\}^k$, let $\mathbf{M}_{\overline{a}} = (\{0, \ldots, \log(n-1) - 1\}, \in, P_1^n, \ldots, P_k^n)$, where $P_i^n = \{m : m \in a_i\}$. Since R is a built-in relation computable in polylogarithmic-time, the language

$$\{1^n \# \mathrm{b}_n(a_1) \# \ldots \# \mathrm{b}_n(a_k) : (a_1, \ldots, a_k) \in R_n,\ n \geq 1\}$$

is decidable in polylogarithmic-time on inputs of the appropriate form. A simple unpadding argument shows then that the language $\{\langle \mathbf{M}_{\overline{a}} \rangle : \overline{a} \in \bigcup_{n \geq 1} R_n\}$ is in $\mathbf{P}$ (the length of $\langle \mathbf{M}_{\overline{a}} \rangle$ is logarithmic in the length of $\langle \{0, \ldots, n-1\}, a_1, \ldots, a_k \rangle$). Hence, by the Immerman-Vardi Theorem, the boolean query $Q = \{\mathbf{M}_{\overline{a}} : \overline{a} \in \bigcup_{n \geq 1} R_n\}$ is definable in least fixed-point logic on the class of all structures of the form $\mathbf{M}_{\overline{a}}$. We may even assume that Q is definable by a sentence of the form $(\mathrm{LFP}_{\overline{x}, X} \varphi)(\overline{0})$ in which φ is a first-order formula, and 0 is a constant for zero. Let $\varphi'(y, z, p_1, \ldots, p_k, \overline{x}, X)$ be the first-order formula over the vocabulary $\{\in\}$ that results from the following substitution in φ: replace each occurrence of an atomic formula of the form $P_i(u)$ by $u \in p_i$; replace each atomic formula of the form $X(\overline{u})$ by $X(y, z, p_1, \ldots, p_k, \overline{u})$; and replace each subformula of the form $(\exists u)(\psi)$ by $(\exists u \in y)(\psi') \vee (\exists u \in z)(\psi')$, where ψ' is the result of applying recursively the substitutions. Clearly, φ' is a Δ_0 formula. Moreover, it is not hard to see that for every $a_1, \ldots, a_k \in \{0, \ldots, n-1\}$, we have that $\mathbf{M}_{\overline{a}} \models (\mathrm{LFP}_{\overline{x}, X} \varphi)(\overline{0})$ if and only if

$$(\{0, \ldots, \log(n-1) - 1\}, \in) \models (\mathrm{LFP}_{y, z, \overline{p}, \overline{x}, X} \varphi')(r, s, a_1, \ldots, a_k, \overline{0}),$$

where s is the largest power of two in the universe, and $r = s - 1$ (observe that the binary representations of s and r are dual words; that is, $j \in s$ if and only if $j \notin r$ for every $j \leq \log(\log(n-1) - 1)$). Since r and s are first-order definable with $\in$, we have shown that R is uniformly definable on $\mathcal{BFR}$ by a sentence of $\mathrm{FO} + \mathrm{LFP}(\Delta_0)$.

It remains to see that (iii) implies (i). Let $\varphi(x_1, \ldots, x_s)$ be a formula witnessing that Q is definable in $\mathrm{FO} + \mathrm{LFP}(\Delta_0)$. Without loss of generality, we may

assume the following normal form for φ:

$$(Q_1 y_1)\cdots(Q_r y_r)(\bigwedge_{i=1}^{s}(\psi_{i,1}\vee\ldots\vee\psi_{i,t}\vee\neg\psi_{i,t+1}\vee\neg\psi_{i,u})),$$

where each Q_i is $\exists$ or $\forall$, and each $\psi_{i,j}$ is either an atomic formula, or a formula of the form $(\mathrm{LFP}_{\overline{z},Z}\theta)(z_1,\ldots,z_s)$, with θ a Δ_0 formula. For every i,j, let $Q_{i,j}$ be the query on $\mathcal{BFR}$ defined by $\psi_{i,j}$, and let A be the following language over the alphabet $\{0,1,\#\}$:

$$\{n\#\mathrm{b}_n(b_1)\#\ldots\#\mathrm{b}_n(b_s)\#i\#j : (b_1,\ldots,b_s)\in Q_{i,j}(\mathbf{BIT}_n)\}.$$

This language will be our oracle set (that it belongs to **P** will be shown later). An alternating Turing machine with oracle A may simulate φ as indicated next. On input $\langle\mathbf{BIT}_n,a_1,\ldots,a_k\rangle$ where $a_1,\ldots,a_k\in\{0,\ldots,n-1\}$, the machine behaves as follows. First, it computes n. To this end, it existentially guesses the position of the leftmost # in the input, and universally branches to check that every smaller position contains a symbol other than #. Then, following the alternation pattern of the quantifier prefix of φ, the machine existentially or universally guesses r words $w_1,\ldots,w_r$ of length $\log(n-1)$ each. The i-th word w_i is meant to be the binary representation of an element $b_i\in\{0,\ldots,n-1\}$ that is to interpret the first-order variable y_i. The machine proceeds then to evaluate each atomic formula $\psi_{i,j}$ as follows. Assume $\psi_{i,j}=\psi_{i,j}(z_1,\ldots,z_s)$, where each variable z_k is either an x_l or a y_l. The machine will write an oracle query of the form $n\#d_1\#\ldots\#d_s\#i\#j$, where $d_k=\mathrm{b}_n(b_l)$ if $z_k=y_l$, and $d_k=\mathrm{b}_n(a_l)$ if $z_k=x_l$. Observe that the length of this query is $O(\log n)$, and is easy to recover from the input (existentially guess each $\mathrm{b}_n(a_l)$ and universally branch to check that all guesses match the input). Clearly, the answer to this query is yes if and only if $\mathbf{BIT}_n\models\psi_{i,j}[d_1,\ldots,d_s]$ by the definition of the oracle set A.

All it remains to show is that the language A belongs to **P**. This is fairly easy. If $\psi_{i,j}$ is an atomic formula, there is almost nothing to see: equalities are checked at once, and atomic formulas of the form $z_i\in z_j$ are also straightforward to check. If $\psi_{i,j}$ is a formula of the form $(\mathrm{LFP}_{\overline{z},Z}\theta)(z_1,\ldots,z_s)$ with θ being a Δ_0 formula, then the query it defines is computable in **DPOLYLOGTIME** on $\mathcal{BFR}$ by Lemma 2. Therefore, since the length of $n\#d_1\#\ldots\#d_s\#i\#j$ is logarithmic in the length of $\langle\mathbf{BIT}_n,d_1,\ldots,d_s\rangle$, a simple unpadding argument puts A in **P**. □

As a corollary, we obtain a characterization of the question on whether all polynomial-time decidable languages are rudimentary. The relationship between **P** and **RUD** remains unknown. It is known however that **NL** $\subseteq$ **RUD** [27], where **NL** is the class of languages accepted in nondeterministic logarithmic-space.

Corollary 1. *The following are equivalent:*

1. $\mathrm{FO} + \mathrm{LFP}(\Delta_0) \subseteq \mathrm{FO}$ *on* $\mathcal{BFR}$,
2. $\mathbf{P} \subseteq \mathbf{RUD}$,
3. $\mathbf{P} \subseteq \mathbf{LINH}$.

Proof: Since $\mathbf{RUD} = \mathbf{LINH} = \mathrm{ATIME}(O(n), O(1))$, it is enough to show that (i) and (iii) are equivalent. The implication from (i) to (iii) follows from Theorem 1 in [3]. For the other implication, assume that $\mathbf{P} \subseteq \mathbf{LINH}$, and let Q be a query on $\mathcal{BFR}$ that is definable by a $\mathrm{FO} + \mathrm{LFP}(\Delta_0)$ formula. By Theorem 2 we have Q is computable in $\mathbf{LH}^{\mathbf{P}}$, and so in $\mathbf{LH}^{\mathbf{LINH}}$ by hypothesis. Let M be an oracle alternating Turing machine witnessing that Q is computable in $\mathbf{LH}^A$ for some $A \in \mathbf{LINH}$, and let N be an alternating Turing machine witnessing that $A \in \mathbf{LINH}$. Since an oracle Turing machine running in logarithmic-time can only ask logarithmically long queries, oracle queries of M may be answered by N in logarithmic-time with respect to the input to M. The number of alternations being constant, it follows that Q is computable in $\mathbf{LH}$. Hence, Q is first-order definable on $\mathcal{BFR}$. □

5 The Presence of Input Predicates

The natural question at this point is what happens when input predicates, in addition to the membership (BIT) relation, are available. That is, we fix a relational vocabulary σ, and we wonder what is captured by $\mathrm{FO} + \mathrm{LFP}(\Delta_0)$ on classes of finite structures over σ with built-in membership relation. Somewhat surprisingly, we are only able to provide an exact answer in the case that σ is a unary vocabulary. In that case, the $\mathrm{LFP}(\Delta_0)$-definable queries still only depend on $O(\log n)$ bits of the input, and a similar argument as before goes through.

Theorem 3. *Let σ be a unary vocabulary, let C be a class of finite structures over σ with built-in membership relation, and let Q be a query on C. Then, the following are equivalent:*

1. *Q is computable in $\mathbf{LH}^{\mathbf{P}}$ on C,*
2. *Q is computable in* **DPOLYLOGTIME***-uniform $\mathbf{AC}^0$ on C,*
3. *Q is definable in $\mathrm{FO} + \mathrm{LFP}(\Delta_0)$ on C.*

Proof: The proofs that (i) implies (ii), and that (ii) implies (iii), go through as in Theorem 2 essentially without change. The proof that (iii) implies (i) uses an argument similar to the one in the proof of Lemma 2. Recall from Lemma 1 that if φ is a Δ_0 formula, then $\mathbf{M} \models \varphi[a_1, \ldots, a_s, A]$ if and only if $\mathbf{M} \models \varphi[a_1, \ldots, a_s, A \cap (\bigcup\{\mathrm{RTC}(a_i) : i \in F(\varphi)\})^k]$. Iterated application of this lemma with each of the relation symbols of σ shows then that $\mathbf{M} \models \varphi[a_1, \ldots, a_s, A]$ if and only if

$$\mathbf{M} \cap B \models \varphi[a_1, \ldots, a_s, A \cap B^k],$$

where $B = \bigcup\{\mathrm{RTC}(a_i) : i \in F(\varphi)\}$, and $\mathbf{M} \cap B$ is the substructure of $\mathbf{M}$ generated by B. In turn, we remark that $B \subseteq \{0, \ldots, \log(|M| - 1) - 1\} \cup \{a_1, \ldots, a_s\}$

since each element of $\mathrm{TC}(a_i)$ is a *bit* position of an element in $\{0,\ldots,|M|-1\}$. Moreover, a straightforward argument reveals that $\mathbf{M}\cap B'$ is an end-extension of $\mathbf{M}\cap B$, where $B'=\{0,\ldots,\log(|M|-1)-1\}\cup\{a_1,\ldots,a_s\}$. Hence, $\mathbf{M}\models\varphi[a_1,\ldots,a_s,A]$ if and only if $\mathbf{M}\cap B\models\varphi[a_1,\ldots,a_s,A\cap B^k]$, and by absoluteness, if and only if $\mathbf{M}\cap B'\models\varphi[a_1,\ldots,a_s,A\cap B'^k]$. With these observations in hand, we claim that:

Claim. If Q is definable in $\mathrm{FO}+\mathrm{LFP}(\Delta_0)$ on C, then Q is definable by a formula of $\mathrm{FO}+\mathrm{LFP}(\Delta_0)$ in which no relation symbol from σ appears within the scope of a fixed-point operator.

Proof: The main idea is that since every $\mathrm{LFP}(\Delta_0)$ formula will only depend on $O(\log n)$ bits of the input predicates by the remarks above (here is the crucial point where we use the fact that the vocabulary is unary), we can existentially quantify these bits outside the $\mathrm{LFP}(\Delta_0)$-formula, and pass them to it as input variables. Formally, the argument is as follows. Assume for simplicity that σ consists of a unique relation symbol R; the general case is as easy. Let φ be a formula defining Q on C. Replace each occurrence in φ of a subformula of the form $(\mathrm{LFP}_{\overline{x},X}\theta)(x_1,\ldots,x_k)$ with θ a Δ_0 formula, by the formula

$$(\exists v)((r\in v\leftrightarrow R(r))\wedge(\forall z\in s)(z\in v\leftrightarrow R(z))\wedge \bigvee_{w\in\{0,1\}^k}\Big(\bigwedge_{w_i=1}R(x_i)\wedge\bigwedge_{w_i=0}\neg R(x_i)\wedge(\mathrm{LFP}_{v,\overline{x},X'}\theta^w)(v,\overline{x})\Big)),$$

where θ^w is the result of replacing each atomic formula of the form $R(u)$, with u a bound variable, by $u\in v$, each atomic formula of the form $R(x_i)$ by $x_i=x_i$, if $w_i=1$, each atomic formula of the form $R(x_j)$ by $x_j\neq x_j$, if $w_j=0$, and each atomic formula of the form $X(\overline{u})$ by $X'(v,\overline{u})$. Here, r and s are existentially quantified variables set to the largest power of two of the universe, and $r-1$ respectively (observe that the binary representations of r and s are dual words). Observe that if v is a witness for the first-order variable of this formula, then its binary representation is encoding the first $\log(n-1)$ bits of R. By the remarks preceding the claim, it is straightforward to check using standard absoluteness arguments that the modified formula is defining Q on C, as required.

The rest of the proof that (iii) implies (i) is now almost identical to the proof of Theorem 2. Namely, access to the input predicates is only required when simulating the first-order part of the formula, and the simulation of the $\mathrm{LFP}(\Delta_0)$-parts of the formula may be asked to an oracle set in $\mathbf{P}$. □

Observe that the argument of Theorem 3 does not go through for vocabularies of higher arities. In the case of digraphs, for example, the reason is that there are $O((\log|M|)^2)$ significant bits (instead of $O(\log|M|)$) in the substructure $\mathbf{M}\cap\{0,\ldots,\log(|M|-1)\}$ of any digraph $\mathbf{M}$. Although we do not provide with an exact characterization of $\mathrm{FO}+\mathrm{LFP}(\Delta_0)$ for vocabularies of higher arities, we are able to compare the expressive power of $\mathrm{FO}+\mathrm{LFP}(\Delta_0)$ with a familiar complexity class. Recall from the introduction that $\mathbf{RUD}_{n^{1/r}}=\mathrm{ATIME}(O(n^{1/r}),O(1))$ (see Corollary 5 in [1]).

Theorem 4. *Let σ be a relational vocabulary of maximum arity r, and let C be the class of all finite structures over σ with built-in membership relation. If $\mathbf{P} \subseteq \mathbf{RUD}_{n^{1/r}}$, then* $\mathrm{FO} + \mathrm{LFP}(\Delta_0) \subseteq \mathrm{FO}$ *on C.*

Proof sketch: Assume $\mathbf{P} \subseteq \mathbf{RUD}_{n^{1/r}}$, and let Q be a query on C definable in FO+LFP(Δ_0). It is enough to show that Q is computable in **LH** on C. Even easier, it is enough to show that each FO+LFP(Δ_0)-formula can be evaluated in **LH** on the appropriate inputs. Let $\varphi(x_1, \ldots, x_k)$ be such a formula. Lemma 2 says that deciding whether $\mathbf{M} \models \varphi[a_1, \ldots, a_k]$ can be done in polylogarithmic-time in $|M|$. Moreover, the same absoluteness argument as in the proof of Theorem 3 reveals that $\mathbf{M} \models \varphi[a_1, \ldots, a_k]$ if and only if $\mathbf{M} \cap B' \models \varphi[a_1, \ldots, a_k]$, where $B' = \{0, \ldots, \log(|M| - 1) - 1\} \cup \{a_1, \ldots, a_k\}$. Since only $O((\log |M|)^r)$ bits are relevant in $\mathbf{M} \cap B'$, the same computation can can be carried over an unpadded input that only contains these bits. The computation time is now polynomial in the length of the (unpadded) input, and therefore, by hypothesis, the same language is decidable in $\mathrm{ATIME}(O(n^{1/r}), O(1)) = \mathbf{RUD}_{n^{1/r}}$ on the appropriate inputs. Since the length of these inputs is $O((\log |M|)^r)$, the alternating computation can be carried over the original inputs in time

$$O(((\log |M|)^r)^{1/r}) = O(\log |M|),$$

and still a constant number of alternations. That is, on the original inputs, the evaluation of φ can be done in **LH** as required. □

As mentioned in the introduction, Theorem 3 sets the link to an important problem related to the uniformity of circuits for integer division. Beame, Cook, and Hoover [7] showed that the problem of dividing two numbers can be computed by **P**-uniform bounded fan-in, logarithmic-depth circuits ($\mathbf{NC}^1$). The result was improved by Reif [28] (see also [19]) who showed that the problem could be computed by **P**-uniform unbounded fan-in, bounded-depth circuits with majority gates ($\mathbf{TC}^0$). However, it is not known whether the uniformity condition of their algorithm can be relaxed to **DLOGTIME**-uniformity, as it is the case for the $\mathbf{TC}^0$ circuits for addition, subtraction, and multiplication (see Barrington, Immerman and Straubing [5]).

On the other hand, it is known that majority gates of polylogarithmically-many bits may be simulated by **DLOGTIME**-uniform $\mathbf{AC}^0$ circuits (see [34] for a similar construction). A circuit $\mathrm{TH}_k(x_1, \ldots, x_m)$ computing whether at least k of the input bits $x_1, \ldots, x_m$ is recursively built as follows:

$$\mathrm{TH}_k(x_1, \ldots, x_m) := \bigvee_{\substack{i_1 + \ldots + i_s \geq k \\ i_j \leq m/s}} \bigwedge_{j=1}^{s} \mathrm{TH}_{i_j}(x_{(j-1)m/s+1}, \ldots, x_{jm/s}),$$

where $m = (\log n)^{O(1)}$, and s is suitably chosen so that the size of the circuit is polynomial in n, and the depth is a constant independent of n (the choice $s = (\log n)^{\epsilon}$ works for sufficiently small ϵ). It is not hard to see that these circuits are **DLOGTIME**-uniform (a clever numbering of gates will tell all the

required information to the **DLOGTIME** algorithm that computes the direct connection language). The well-known power of $\mathbf{AC}^0$ circuits to do arithmetic on numbers with polylogarithmically-many significant bits follows from Reif's result, and the known algorithms for addition, subtraction and multiplication. However, while addition, subtraction and multiplication of polylogarithmically-long numbers admit **DLOGTIME**-uniform $\mathbf{AC}^0$ such circuits, the known algorithms for division fall short since they only give **DPOLYLOGTIME**-uniform $\mathbf{AC}^0$ circuits. We note that Theorem 3 implies that division of numbers with polylogarithmically-many significant bits is definable in $\mathrm{FO} + \mathrm{LFP}(\Delta_0)$ on the class of finite words with built-in membership relation. We do not know, however, of a direct proof of this fact.

Acknowledgments

I am grateful to José L. Balcázar and Phokion Kolaitis for insightful comments, and to Ricard Gavaldà for teaching me about the simulation of $\mathbf{TC}^0$ circuits of polylogarithmically many bits by $\mathbf{AC}^0$ circuits. I am also grateful to Martin Grohe who asked the question that led to Theorem 3.

References

[1] E. Allender and V. Gore. Rudimentary reductions revisited. *Information Processing Letters*, 40:89–95, 1991.
[2] E. Allender and V. Gore. A uniform circuit lower bound for the permanent. *SIAM Journal of Computing*, 23(5):1026–1049, 1994.
[3] A. Atserias and Ph. G. Kolaitis. First-order logic vs. fixed-point logic in finite set theory. In *14th IEEE Symposium on Logic in Computer Science*, pages 275–284, 1999.
[4] D. M. Barrington and N. Immerman. Time, hardware, and uniformity. In *Complexity Theory Retrospective II*, pages 1–22. Springer-Verlag, 1997.
[5] D.M. Barrington, N. Immerman, and H. Straubing. On uniformity within NC^1. *Journal of Computer and System Sciences*, 41(3):274–306, 1990.
[6] J. Barwise. *Admissible Sets and Structures*. Springer-Verlag, 1975.
[7] P. W. Beame, S. A. Cook, and H. J. Hoover. Log depth circuits for division and related problems. *SIAM Journal of Computing*, 15(4):994–1003, 1986.
[8] J. H. Bennett. *On Spectra*. PhD thesis, Princeton University, 1962.
[9] S. R. Buss. The boolean function value problem is in ALOGTIME. In *28th Annual IEEE Symposium on Foundations of Computer Science*, pages 123–131, 1987.
[10] A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *Journal of the ACM*, 28:114–133, 1981.
[11] A. Dawar, K. Doets, S. Lindell, and S. Weinstein. Elementary properties of finite ranks. *Mathematical Logic Quarterly*, 44:349–353, 1998.
[12] A. Dawar and L. Hella. The expressive power of finitely many generalized quantifiers. *Information and Computation*, 123:172–184, 1995.
[13] A. Dawar, S. Lindell, and S. Weinstein. First order logic, fixed point logic and linear order. In *Computer Science Logic '95*, volume 1092 of *Lecture Notes in Computer Science*, pages 161–177. Springer-Verlag, 1996.

[14] L. Fortnow. Time-space tradeoffs for satisfiability. In *12th IEEE Conference in Computational Complexity*, pages 52–60, 1997. To appear in *Journal of Computer and System Sciences*.

[15] Y. Gurevich, N. Immerman, and S. Shelah. McColm's conjecture. In *9th IEEE Symposium on Logic in Computer Science*, pages 10–19, 1994.

[16] Y. Gurevich and S. Shelah. Fixed-point extensions of first-order logic. *Annals of Pure and Applied Logic*, 32(3):265–280, 1986.

[17] N. Immerman. Relational queries computable in polynomial time. *Information and Computation*, 68:86–104, 1986.

[18] N. Immerman. Expressibility and parallel complexity. *SIAM Journal of Computing*, 18:625–638, 1989.

[19] N. Immerman and S. Landau. The complexity of iterated multiplication. *Information and Computation*, 116(1):103–116, 1995.

[20] N. Jones. Context-free languages and rudimentary attributes. *Mathematical Systems Theory*, 3:102–109, 1969.

[21] N. D. Jones. Space-bounded reducibility among combinatorial problems. *Journal of Computer and System Sciences*, 11:68–85, 1975. Corrigendum: *Journal of Computer and System Sciences* 15:241, 1977.

[22] Ph. G. Kolaitis and M. Y. Vardi. Fixpoint logic vs. infinitary logic in finite-model theory. In *7th IEEE Symposium on Logic in Computer Science*, pages 46–57, 1992.

[23] S. Lindell. A purely logical characterization of circuit uniformity. In *7th IEEE Structure in Complexity Theory*, pages 185–192, 1992.

[24] R. J. Lipton and A. Viglas. On the complexity of SAT. In *40th Annual IEEE Symposium on Foundations of Computer Science*, pages 459–464, 1999.

[25] J. A. Makowsky. Invariant definability and P/poly. To appear in Lecture Notes in Computer Science, Proceedings of Computer Science Logic 1998, 1999.

[26] Y. N. Moschovakis. *Elementary Induction on Abstract Structures*. North-Holland, 1974.

[27] V. A. Nepomnjascii. Rudimentary predicates and Turing calculations. *Soviet Math. Dokl.*, 11:1462–1465, 1970.

[28] J. H. Reif. On threshold circuits and polynomial computation. In *2nd IEEE Structure in Complexity Theory*, pages 118–123, 1987.

[29] W. L. Ruzzo. On uniform circuit complexity. *Journal of Computer and System Sciences*, 22:365–383, 1981.

[30] V. Y. Sazonov. On bounded set theory. In *Logic and Scientific Methods*, pages 85–103. Kluwer Academic Publishers, 1997.

[31] M. Sipser. Borel sets and circuit complexity. In *15th Annual ACM Symposium on the Theory of Computing*, pages 61–69, 1983.

[32] R. Smullyan. Theory of formal systems. In *Annals of Mathematics Studies*, volume 47. Princeton University Press, 1961.

[33] L. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3:1–22, 1977.

[34] I. Wegener. *The Complexity of Boolean Functions*, pages 243–247. John Wiley & Sons, 1987.

[35] C. Wrathall. Rudimentary predicates and relative computation. *SIAM Journal of Computing*, 7(2):194–209, 1978.

Hypersequents and the Proof Theory of Intuitionistic Fuzzy Logic*

Matthias Baaz[1] and Richard Zach[2]

[1] Institut für Algebra und Computermathematik E118.2,
Technische Universität Wien, A–1040 Vienna, Austria, baaz@logic.at
[2] Institut für Computersprachen E185.2,
Technische Universität Wien, A–1040 Vienna, Austria, zach@logic.at

Abstract. Takeuti and Titani have introduced and investigated a logic they called intuitionistic fuzzy logic. This logic is characterized as the first-order Gödel logic based on the truth value set $[0, 1]$. The logic is known to be axiomatizable, but no deduction system amenable to proof-theoretic, and hence, computational treatment, has been known. Such a system is presented here, based on previous work on hypersequent calculi for propositional Gödel logics by Avron. It is shown that the system is sound and complete, and allows cut-elimination. A question by Takano regarding the eliminability of the Takeuti-Titani density rule is answered affirmatively.

1 Introduction

Intuitionistic fuzzy logic **IF** was originally defined by Takeuti and Titani to be the logic of the complete Heyting algebra $[0, 1]$. In standard many-valued terminology, **IF** is $[0, 1]$-valued first-order Gödel logic, with truth functions as defined below. The finite-valued propositional versions of this logic were introduced by Gödel [8], and have spawned a sizeable area of logical research subsumed under the title "intermediate logics" (intermediate between classical and intuitionistic logic). The infinite-valued propositional Gödel logic was studied by Dummett [6], who showed that it is axiomatized by **LC**, i.e., intuitionistic propositional logic plus the linearity axiom $(A \supset B) \vee (B \supset A)$.

Takeuti and Titani [13] characterized **IF** by a calculus which extends the intuitionistic predicate calculus **LJ** by several axioms as well as the density rule

$$\frac{\Gamma \vdash A \vee (C \supset p) \vee (p \supset B)}{\Gamma \vdash A \vee (C \supset B)}\ tt'$$

This rule can be read as expressing the fact that the set of truth values is densely ordered. In this sense, the Takeuti-Titani axiomatization is the natural axiomatization of the $[0, 1]$-valued Gödel logic. The valid formulas of **IF** are

2000 Mathematics Subject Classification: Primary 03B50; Secondary 03B55, 03F05.
* Research supported by the Austrian Science Fund under grant P–12652 MAT

P. Clote and H. Schwichtenberg (Eds.): CSL 2000, LNCS 1862, pp. 187–201, 2000.

also characterized as those formulas valid in *every* first-order Gödel logic based on a linearly ordered set of truth-values (this is obvious for all logics based on truth value sets $\subseteq [0,1]$, since a countermodel in such a truth-value set can be straightforwardly embedded in $[0,1]$. The general claim was established by Horn [10]). In this characterization, the density rule is not a natural assumption, since not every linearly ordered truth-value set is densely ordered. It follows from this characterization that the density rule is redundant for the axiomatization of **IF**, and completeness proofs without it have been given by Horn [10] and Takano [11].[1] Takano posed the question of whether a syntactic elimination of the density rule is also possible.

More recently, another axiomatizable first-order extension of **LC** has been studied by Corsi [4, 5] and Avellone et al. [1]. This extension is defined not via many-valued semantics but as the class of formulas valid in all linearly ordered intuitionistic Kripke models. It is different from **IF**; specifically, the formula $(\vee\forall)$ below is not valid in it. **IF** can, however, also be characterized as the set of formulas valid in all linearly ordered Kripke models with constant domains (this was first observed by Gabbay [7, §3]).

The interest of **IF** lies in the fact that it combines properties of logics for approximate reasoning with properties of intuitionistic logic. On the one hand, **IF** is one of the basic t-norm logics (see Hájek [9]), on the other, it is an extension of intuitionistic logic which corresponds to concurrency (as has been argued by Avron [2]). We present here a calculus for **IF** which is adequate for further proof-theoretic study. The basic result in this regard is the cut-elimination theorem for this calculus, from which a midhypersequent-theorem can be derived. This theorem, in turn, corresponds to Herbrand's Theorem in classical logic, and as such is a possible basis for automated theorem proving in **IF**.

The calculus also allows us to investigate the proof-theoretic effects of the Takeuti-Titani rule. We give a positive answer to Takano's question, showing that the density rule can be eliminated from **IF**-proofs. A simple example illustrates the possible structural differences between proofs with and without the Takeuti-Titani rule.

2 Syntax and Semantics of Intuitionistic Fuzzy Logic

The language L of **IF** is a usual first-order language with propositional variables and where free (a, b, ...) and bound (x, y, ...) variables are distinguished.

Definition 1. An **IF**-*interpretation* $\Im = \langle D, \mathbf{s}\rangle$ is given by the *domain* D and the *valuation function* $\mathbf{s}$. Let L^D be L extended by constants for each element of D. Then $\mathbf{s}$ maps atomic formulas in $\mathrm{Frm}(L^D)$ into $[0,1]$, $d \in D$ to itself, n-ary function symbols to functions from D^n to D, and free variables to elements of D.

The valuation function $\mathbf{s}$ can be extended in the obvious way to a function on all terms. The valuation for formulas is defined as follows:

[1] Note that the corresponding axiom $(\forall p)((A \supset p) \vee (p \supset B)) \supset (A \supset B)$ is not redundant in quantified *propositional* $[0,1]$-valued Gödel logic. See [3].

1. $A \equiv P(t_1, \ldots, t_n)$ is atomic: $\Im(A) = \mathbf{s}(P)(\mathbf{s}(t_1), \ldots, \mathbf{s}(t_n))$.
2. $A \equiv \neg B$:
$$\Im(\neg B) = \begin{cases} 0 & \text{if } \Im(B) \neq 0 \\ 1 & \text{otherwise.} \end{cases}$$
3. $A \equiv B \wedge C$: $\Im(B \wedge C) = \min(\Im(B), \Im(C))$.
4. $A \equiv B \vee C$: $\Im(B \vee C) = \max(\Im(A), \Im(B))$.
5. $A \equiv B \supset C$:
$$\Im(B \supset C) = \begin{cases} \Im(C) & \text{if } \Im(B) > \Im(C) \\ 1 & \text{if } \Im(B) \leq \Im(C). \end{cases}$$

The set $\mathrm{Distr}_{\Im}(A(x)) = \{\Im(A(d)) : d \in D\}$ is called the *distribution* of $A(x)$. The quantifiers are, as usual, defined by infimum and supremum of their distributions.

(6) $A \equiv (\forall x)B(x)$: $\Im(A) = \inf \mathrm{Distr}_{\Im}(B(x))$.
(7) $A \equiv (\exists x)B(x)$: $\Im(A) = \sup \mathrm{Distr}_{\Im}(B(x))$.

$\Im$ *satisfies* a formula A, $\Im \models A$, if $\Im(A) = 1$. A formula A is **IF**-valid if every **IF**-interpretation satisfies it.

Note that, as in intuitionistic logic, $\neg A$ may be defined as $A \supset \bot$, where $\bot$ is some formula that always takes the value 0.

3 Hypersequents and IF

Takeuti and Titani's system **IF** is based on Gentzen's sequent calculus **LJ** for intuitionistic logic with a number of extra axioms

$$\vdash (A \supset B) \vee ((A \supset B) \supset B) \qquad \text{(Ax1)}$$
$$(A \supset B) \supset B \vdash (B \supset A) \vee B \qquad \text{(Ax2)}$$
$$(A \wedge B) \supset C \vdash (A \supset C) \vee (B \supset C) \qquad \text{(Ax3)}$$
$$(A \supset (B \vee C)) \vdash (A \supset B) \vee (A \supset C) \qquad \text{(Ax4)}$$
$$(\forall x)(A(x) \vee B) \vdash (\forall x)A(x) \vee B \qquad (\vee\forall)$$
$$(\forall x)A(x) \supset C \vdash (\exists x)(A(x) \supset D) \vee (D \supset C) \qquad (\forall \supset)$$

(where x does not occur in B or D) and the following additional inference rule:

$$\frac{\Gamma \vdash A \vee (C \supset p) \vee (p \supset B)}{\Gamma \vdash A \vee (C \supset B)}\ tt'$$

where p is a propositional eigenvariable (i.e., it does not occur in the lower sequent). It is known that the extra inference rule is redundant. In fact, the system **H** of Horn [10] consisting of **LJ** plus the schemata

$$(\forall x)(A(x) \vee B) \supset (\forall x)A(x) \vee B \qquad (\vee\forall)$$
$$(A \supset B) \vee (B \supset A) \qquad (D)$$

is complete for **IF** (see also [11]). Neither of these systems, however, has decent proof-theoretic properties such as cut elimination, nor is a syntactic method for

the elimination of the Takeuti-Titani rule (tt') known. Takano [11] has posed the question of a syntactic elimination procedure of the Takeuti-Titani rule as an open problem.

We present a system which has the required properties, and which allows the syntactic elimination of the Takeuti-Titani rule. Our system is based on Avron's [2] cut-free axiomatization of **LC** using a hypersequent calculus.

Definition 2. A *sequent* is an expression of the form

$$\Gamma \vdash \Delta$$

where Γ and Δ are finite multisets of formulas, and Δ contains at most one formula. A *hypersequent* is a finite multiset of sequents, written as

$$\Gamma_1 \vdash \Delta_1 \mid \ldots \mid \Gamma_n \vdash \Delta_n$$

The hypersequent calculus **HIF** has the following axioms and rules:
Axioms: $A \vdash A$, for any formula A.
Internal structural rules:

$$\frac{G \mid \Gamma \vdash \Delta}{G \mid A, \Gamma \vdash \Delta}\, iw\vdash \qquad \frac{G \mid \Gamma \vdash}{G \mid \Gamma \vdash A}\, \vdash iw \qquad \frac{G \mid A, A, \Gamma \vdash \Delta}{G \mid A, \Gamma \vdash \Delta}\, ic\vdash$$

External structural rules:

$$\frac{G}{G \mid \Gamma \vdash \Delta}\, ew \qquad \frac{G \mid \Gamma \vdash \Delta \mid \Gamma \vdash \Delta}{G \mid \Gamma \vdash \Delta}\, ec$$

Logical rules:

$$\frac{G \mid \Gamma \vdash A}{G \mid \neg A, \Gamma \vdash}\, \neg\vdash \qquad \frac{G \mid A, \Gamma \vdash}{G \mid \Gamma \vdash \neg A}\, \vdash\neg$$

$$\frac{G \mid A, \Gamma \vdash \Delta \quad G \mid B, \Gamma \vdash \Delta}{G \mid A \vee B, \Gamma \vdash \Delta}\, \vee\vdash \qquad \frac{G \mid \Gamma \vdash A \quad G \mid \Gamma \vdash B}{G \mid \Gamma \vdash A \wedge B}\, \vdash\wedge$$

$$\frac{G \mid \Gamma \vdash A}{G \mid \Gamma \vdash A \vee B}\, \vdash\vee_1 \qquad \frac{G \mid A, \Gamma \vdash \Delta}{G \mid A \wedge B, \Gamma \vdash \Delta}\, \wedge\vdash_1$$

$$\frac{G \mid \Gamma \vdash B}{G \mid \Gamma \vdash A \vee B}\, \vdash\vee_2 \qquad \frac{G \mid B, \Gamma \vdash \Delta}{G \mid A \wedge B, \Gamma \vdash \Delta}\, \wedge\vdash_2$$

$$\frac{G \mid \Gamma_1 \vdash A \quad G \mid B, \Gamma_2 \vdash \Delta}{G \mid A \supset B, \Gamma_1, \Gamma_2 \vdash \Delta}\, \supset\vdash \qquad \frac{G \mid A, \Gamma \vdash B}{G \mid \Gamma \vdash A \supset B}\, \vdash\supset$$

$$\frac{G \mid A(t), \Gamma \vdash \Delta}{G \mid (\forall x)A(x), \Gamma \vdash \Delta}\, \forall\vdash \qquad \frac{G \mid \Gamma \vdash A(a)}{G \mid \Gamma \vdash (\forall x)A(x)}\, \vdash\forall$$

$$\frac{G \mid A(a), \Gamma \vdash \Delta}{G \mid (\exists x)A(x), \Gamma \vdash \Delta}\, \exists\vdash \qquad \frac{G \mid \Gamma \vdash A(t)}{G \mid \Gamma \vdash (\exists x)A(x)}\, \vdash\exists$$

Cut:

$$\frac{G \mid \Gamma \vdash A \quad G \mid A, \Pi \vdash \Lambda}{G \mid \Gamma, \Pi \vdash \Lambda}\, cut$$

Communication:

$$\frac{G \mid \Theta_1, \Theta_1' \vdash \Xi_1 \quad G \mid \Theta_2, \Theta_2' \vdash \Xi_2}{G \mid \Theta_1, \Theta_2' \vdash \Xi_1 \mid \Theta_1', \Theta_2 \vdash \Xi_2}\ cm$$

Density:

$$\frac{G \mid \Phi \vdash p \mid p, \Psi \vdash \Sigma}{G \mid \Phi, \Psi \vdash \Sigma}\ tt$$

The rules $(\vdash \forall)$, $(\exists \vdash)$, and (tt) are subject to eigenvariable conditions: the free variable a and the propositional variable p, respectively, must not occur in the lower hypersequent. We denote the calculus obtained from **HIF** by omitting the cut rule by $\mathbf{HIF}^-$, and that obtained by omitting (tt) by $\mathbf{HIF}^*$.

The semantics of **IF** can easily be extended to hypersequents by mapping a hypersequent H

$$\Gamma_1 \vdash \Delta_1 \mid \ldots \mid \Gamma_n \vdash \Delta_n$$

to the formula H^*

$$(\bigwedge \Gamma_1 \supset \bigvee \Delta_1) \vee \ldots \vee (\bigwedge \Gamma_n \supset \bigvee \Delta_n)$$

where $\bigwedge \Gamma_i$ denotes the conjunction of the formulas in Γ_i or $\top$ if Γ_i is empty, and $\bigvee \Delta_i$ the disjunction of the formulas in Δ_i or $\bot$ if Δ_i is empty. Deriving a formula A in **HIF** then is equivalent to deriving the sequent $\vdash A$: the translation of $\vdash A$, i.e., $\top \supset A$ is equivalent to A.

Theorem 3 (Soundness). *Every hypersequent H derivable in* **HIF** *is* **IF***-valid.*

Proof. By induction on the length of the proof. It will suffice to show that the axioms are valid, and that the quantifier rules and (tt) preserve validity.

The soundness of the quantifier rules is established by observing that corresponding quantifier shifting rules are intuitionistically valid. For instance, since

$$(\exists x)(B \vee A(x)) \supset (B \vee (\exists x)A(x)) \qquad (\vee\exists)$$

$$(\exists x)(B \supset A(x)) \supset B \supset (\exists x)A(x) \qquad (\supset\exists)$$

are intuitionistically valid, it is easily seen that $\vdash\exists$ is a sound rule. The only problematic rules are $(\vdash\forall)$ and $(\exists\vdash)$. Suppose $G \mid \Gamma \vdash A(a)$ is derivable in **HIF**. By induction hypothesis, $G^* \vee (\bigwedge \Gamma \supset A(a))$ is valid. Then certainly $(\forall x)(G^* \vee (\bigwedge \Gamma \supset A(x)))$ is **IF**-valid. Since a did not occur in G or Γ, we may now assume that x does not either. Since the quantifier shift $(\vee\forall)$, i.e.,

$$(\forall x)(B \vee A(x)) \supset (B \vee (\forall x)A(x)),$$

is valid in **IF**, we see that $G^* \vee (\forall x)(\bigwedge \Gamma \supset A(x))$ is valid. The result follows since

$$(\forall x)(B \supset A(x)) \supset B \supset (\forall x)A(x)$$

is intuitionistically valid, and hence **IF**-valid.

The communication rule is sound as well. Suppose the interpretation $\Im$ satisfies the premises of (cm). The only case where the conclusion is not obviously also satisfied is if $\Im(\Theta_1') \leq \Im(\Xi_1)$ and $\Im(\Theta_2') \leq \Im(\Xi_2)$. If the left lower sequent is not satisfied, we have $\Im(\Xi_1) < \Im(\Theta_2')$, and hence $\Im(\Theta_1') \leq \Im(\Xi_2)$, and thus the right lower sequent is satisfied. Similarly if the right lower sequent is not satisfied.

For (tt) we may argue as follows: Suppose that the hypersequent

$$H = G \mid \Phi \vdash p \mid p, \Psi \vdash \Sigma$$

is **IF**-valid. Let $\Im$ be an interpretation, and let $\Im_r$ be just like $\Im$ except that $\Im(p) = r$. Since p does not occur in the conclusion hypersequent

$$H' = G \mid \Phi, \Psi \vdash \Sigma$$

we have $\Im(H') = \Im_r(H')$ and $\Im(G) = \Im_r(G)$. If $\Im \models G$ we are done. Otherwise, assume that $\Im \not\models H'$, i.e.,

$$r_1 = \min\{\Im(\Phi), \Im(\Psi)\} > \Im(\Sigma) = r_2$$

Let $r = (r_1 + r_2)/2$. Now consider $\Im_r$: $\Im_r \not\models G$ by assumption; $\Im_r \not\models \Phi \vdash p$, since $\Im_r(\Phi) > r$; and $\Im_r \not\models p, \Psi \vdash \Sigma$, since $\Im_r(\Psi) > r > \Im_r(\Sigma)$. Hence, $\Im_r \not\models H$, a contradiction. □

Theorem 4 (Completeness). *Every* **IF***-valid hypersequent is derivable in* **HIF***.*

Proof. Observe that a hypersequent H and its canonical translation $\vdash H^*$ are interderivable using the cut rule and the following derivable hypersequents

$$A \vee B \vdash A \mid A \vee B \vdash B \qquad A \supset B, A \vdash B$$
$$A \wedge B \vdash A \qquad A \vdash A \vee B$$

Thus it suffices to show that the characteristic axioms of **IF** are derivable; a simple induction on the length of proofs shows that proofs in intuitionistic predicate calculus together with the axioms (D) and $(\vee\forall)$ can be simulated in **HIF**. The formula (D) is easily derivable using the communication rule.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{A \vdash A \quad B \vdash B}{A \vdash B \mid B \vdash A}\, cm}{\vdash A \supset B \mid B \vdash A}\, \vdash\supset}{\vdash A \supset B \mid \ \vdash B \supset A}\, \vdash\supset}{\vdash (A \supset B) \vee (B \supset A) \mid \ \vdash B \supset A}\, \vdash\vee}{\vdash (A \supset B) \vee (B \supset A) \mid \ \vdash (A \supset B) \vee (B \supset A)}\, \vdash\vee}{\vdash (A \supset B) \vee (B \supset A)}\, ec$$

The formula $(\vee\forall)$ can be obtained thus:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{A(a)\vdash A(a)\quad B\vdash B}{B\vdash A(a)\mid A(a)\vdash B}\,cm\quad\dfrac{B\vdash B}{B\vdash A(a)\mid B\vdash B}\,ew}{B\vdash A(a)\mid B\vee A(a)\vdash B}\,\vee\vdash\quad\dfrac{A(a)\vdash A(a)}{A(a)\vdash A(a)\mid B\vee A(a)\vdash B}\,ew}{B\vee A(a)\vdash A(a)\mid B\vee A(a)\vdash B}\,\vee\vdash}{(\forall x)(B\vee A(x))\vdash A(a)\mid B\vee A(a)\vdash B}\,\forall\vdash}{(\forall x)(B\vee A(x))\vdash A(a)\mid(\forall x)(B\vee A(x))\vdash B}\,\forall\vdash}{(\forall x)(B\vee A(x))\vdash(\forall x)A(x)\mid(\forall x)(B\vee A(x))\vdash B}\,\vdash\forall}{(\forall x)(B\vee A(x))\vdash B\vee(\forall x)A(x)}\,\vdash\vee$$

The last line is obtained from the preceding by two $(\vdash\vee)$ inferences, followed by an external contraction. We indicate this with the double inference line. □

Of course, the other axioms of Takeuti's and Titani's system are also derivable. We will leave the propositional axioms 1–4 as an exercise to the reader, and give the derivation on of $(\forall\supset)$ as another example:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{A(a)\vdash A(a)\quad D\vdash D}{A(a)\vdash D\mid D\vdash A(a)}\,cm}{\vdash A(a)\supset D\mid D\vdash A(a)}\,\vdash\supset}{\vdash(\exists x)(A(x)\supset D)\mid D\vdash A(a)}\,\vdash\exists}{\vdash(\exists x)(A(x)\supset D)\mid D\vdash(\forall x)A(x)}\,\vdash\forall\quad\dfrac{C\vdash C}{\vdash(\exists x)(A(x)\supset D)\mid C\vdash C}\,ew}{\vdash(\exists x)(A(x)\supset D)\mid(\forall x)A(x)\supset C,D\vdash C}\,\supset\vdash}{\vdash(\exists x)(A(x)\supset D)\mid(\forall x)A(x)\supset C\vdash D\supset C}\,\vdash\supset}{(\forall x)A(x)\supset C\vdash(\exists x)(A(x)\supset D)\vee(D\supset C)}\,\vdash\vee$$

4 Cut Elimination and Midhypersequent Theorem

Theorem 5 (Cut Elimination). *Any derivation of a hypersequent G in* **HIF** *can be transformed into a derivation of G in* $\mathbf{HIF}^-$.

This theorem is proved in the usual way by induction on the number of applications of the cut rule, using the following lemma.

Lemma 6. *Suppose the hypersequents*

$$H_1 = G\mid\Gamma\vdash A\quad\text{and}\quad H_2 = G\mid\Pi\vdash\Lambda$$

are cut-free derivable. Then

$$H = G\mid\Gamma,\Pi^*\vdash\Lambda$$

where Π^* is obtained from Π by removing all occurrences of A, is cut-free provable, and the number of applications of (ec) in the resulting proof is not more than the sum of applications of (ec) in γ and δ.

Proof. Let γ and δ be the cut-free proofs of G and H, respectively. We may assume, renaming variables if necessary, that the eigenvariables in γ and δ are distinct. The proof follows Gentzen's original *Hauptsatz*. Define the following measures on the pair $\langle \gamma, \delta \rangle$: the *rank* $r = \mathrm{len}(\gamma) + \mathrm{len}(\delta)$, the *degree* $d = \deg(A)$, and the *order* o is the number of applications of the (ec) rule in γ, δ. We proceed by induction on the lexicographical order of $\langle d, o, r \rangle$.

If either H_1 or H_2 is an axiom, then H can be derived from H_1 or H_2, respectively, using only weakenings. (This includes the case where $r = 2$).

Otherwise, we distinguish cases according to the last inferences in γ and δ. The induction hypothesis is that the claim of the lemma is true whenever the degree is $< d$ or is $= d$ and either the order $< o$, or the order $= o$ and the rank $< r$.

(1) γ or δ ends in an inference which acts on a sequent in G. We may invoke the induction hypothesis on the premises of H_1 or H_2, and H_2 or G_2, respectively.

(2) γ or δ ends in (ec). For instance, γ ends in

$$\dfrac{\begin{array}{c}\vdots\ \gamma' \\ G \mid \Gamma \vdash A \mid \Gamma \vdash A\end{array}}{G \mid \Gamma \vdash A}\ ec$$

Apply the induction hypothesis to γ' and δ. The resulting proof γ'' of

$$G \mid \Gamma \vdash A \mid \Gamma, \Pi^* \vdash \Lambda$$

has one less (ec) than γ (although it may be much longer), and so the induction hypothesis applies again to γ'' and δ.

(3) γ or δ end in another structural inference, (tt), or (cm): These cases are unproblematic applications of the induction hypothesis to the premises, followed by applications of structural inferences.

For example, assume γ ends in (cm), i.e.,

$$\dfrac{\begin{array}{c}\vdots\ \gamma_1 \\ G \mid \Theta_1, \Theta_1' \vdash \Xi_1\end{array} \quad \begin{array}{c}\vdots\ \gamma_2 \\ G \mid \Theta_2, \Theta_2' \vdash A\end{array}}{G \mid \Theta_1, \Theta_2' \vdash \Xi_1 \mid \Theta_1', \Theta_2 \vdash A}\ cm$$

where $\Gamma = \Theta_1', \Theta_2$. Apply the deduction hypothesis to the right premise and H_2 to obtain a cut-free proof of

$$G \mid \Theta_2, \Theta_2', \Pi^* \vdash \Lambda$$

Using applications of (ew) and (cm), we obtain the desired result.

The case of (tt) may be of special interest. Suppose γ ends in(tt), with

$$\dfrac{G \mid \Phi \vdash p \mid p, \Psi \vdash A}{G \mid \Phi, \Psi \vdash A}\ tt$$

Apply the induction hypothesis to the premises of H_1 and H_2, and apply (tt) to obtain the desired proof:

$$\frac{G \mid \Phi \vdash p \mid p, \Psi, \Pi^* \vdash \Lambda}{G \mid \Phi, \Psi, \Pi^* \vdash \Lambda}\ tt$$

The case of δ ending in (tt) is handled similarly.

(4) γ ends in a logical inference not involving the cut formula, or δ ends in a logical inference not involving the cut formula. These cases are easily handled by appeal to the induction hypothesis and application of appropriate logical and structural inferences. We outline the case where γ ends in $(\supset\vdash)$:

$$\frac{\begin{array}{cc}\vdots\ \gamma_1 & \vdots\ \gamma_2 \\ G \mid C, \Gamma \vdash A & G \mid \Gamma \vdash B\end{array}}{G \mid B \supset C, \Gamma \vdash A}\ \supset\vdash$$

We apply the induction hypothesis to the left premise and H_2, and apply $(\supset\vdash)$:

$$\frac{G \mid C, \Gamma, \Pi^* \vdash \Lambda \quad G \mid \Gamma \vdash B}{G \mid B \supset C, \Gamma, \Pi^* \vdash \Lambda}$$

(5) Both γ and δ end in logical inferences acting on a cut formula. For instance, if $A = B \supset C$ we have

$$\frac{\vdots\ \gamma_1}{\dfrac{G \mid B, \Gamma \vdash C}{G \mid \Gamma \vdash B \supset C}\vdash\supset} \qquad \frac{\begin{array}{cc}\vdots\ \delta_1 & \vdots\ \delta_2 \\ G \mid \Pi_1 \vdash B & G \mid C, \Pi_2 \vdash \Lambda\end{array}}{G \mid B \supset C, \Pi_1, \Pi_2 \vdash \Lambda}\ \supset\vdash$$

First we find proofs δ_1' and δ_2' of

$$G \mid \Gamma, \Pi_1^* \vdash B \quad \textit{and} \quad G \mid C, \Gamma, \Pi_2^* \vdash \Lambda$$

either by applying the induction hypothesis to γ and δ_1 or δ_2 if Π_1 or Π_2, respectively, contain $B \supset C$, or otherwise by adding (ic)-inferences to δ_1 and δ_2. Now apply the induction hypothesis based on the reduced degree of the cut formulas twice: first to δ_1' and γ_1 to obtain $G \mid \Gamma, \Gamma, \Pi_1^* \vdash C$, and then to the resulting proof and δ_2' to obtain

$$G \mid \Gamma, \Gamma, \Gamma, \Pi_1^*, \Pi_2^* \vdash \Lambda.$$

The desired result follows by several applications of (ic).

The other cases are similar and are left to the reader. □

Cut elimination is a basic prerequisite for proof theoretic and computational treatments of a logic. As an immediate consequence of cut elimination we have the subformula property: every **IF**-valid formula has a proof which only contains subformulas of the endformula (plus possibly propositional variables used in (tt)). Another important corollary is the midhypersequent theorem. It corresponds to Herbrand's Theorem for classical logic and is thus the basis for any resolution-style automated proof method.

Theorem 7. *Any hypersequent H with only prefix formulas has a proof where no propositional inference follows a quantifier inference. Such a proof contains one or more hypersequents M, called midhypersequents, so that M contains no quantifiers, all the inferences above M are propositional or structural, and all the inferences below M are either quantifier inferences of structural inferences.*

Proof. This is proved exactly as for the classical and intuitionistic case (see Takeuti [12]). First, observe that all axioms are cut-free derivable from atomic axioms. The cut-elimination theorem thus provides us with a cut-free proof π of H from atomic axioms. Next, observe that the $(\vee \vdash)$ rule can be simulated without using cuts by the rule

$$\frac{G \mid A, \Gamma \vdash \Delta_1 \quad G \mid B, \Gamma \vdash \Delta_2}{G \mid A \vee B, \Gamma \vdash \Delta_1 \mid A \vee B, \Gamma \vdash \Delta_2} \vee \vdash'$$

The rule can be derived as follows (we omit side sequents):

$$\dfrac{\dfrac{\dfrac{A, \Gamma \vdash \Delta_1 \quad B, \Gamma \vdash \Delta_2}{B, \Gamma \vdash \Delta_1 \mid A, \Gamma \vdash \Delta_2}\, cm \qquad A, \Gamma \vdash \Delta_1}{A \vee B, \Gamma \vdash \Delta_1 \mid A, \Gamma \vdash \Delta_2} \vee \vdash \qquad B, \Gamma \vdash \Delta_2}{A \vee B, \Gamma \vdash \Delta_1 \mid A \vee B, \Gamma \vdash \Delta_2} \vee \vdash$$

Of course, $(\vee \vdash')$ together with (ec) simulates $(\vee \vdash)$. We replace all applications of $(\vee \vdash)$ by applications of $(\vee \vdash')$ in our cut-free proof.

Define the order of a quantifier inference in π to be the number of propositional inferences under it, and the order of π as the sum of the orders of its quantifier inferences. The proof is by induction on the order of π. The only interesting case is of $(\vee \vdash')$ occurring below a quantifier inference, since this case does not work for intuitionistic logic.

Suppose π contains a $(\vdash \forall)$ inference above a $(\vee \vdash')$ inference, and so that all the inferences in between are structural. We have the following situation:

$$\dfrac{\dfrac{\dfrac{\vdots}{G \mid A, \Gamma \vdash \Delta} \quad \dfrac{\dfrac{\vdots\ \delta'}{G' \mid \Gamma' \vdash A(a)}}{G' \mid \Gamma' \vdash (\forall x)A(x)} \vdash \forall}{G \mid A \vee B, \Gamma \vdash \Delta \mid A \vee B, \Gamma \vdash (\forall x)A(x)} \vee \vdash' \qquad \dfrac{\vdots\ \delta}{G \mid B, \Gamma \vdash (\forall x)A(x)}}{G \mid A \vee B, \Gamma \vdash \Delta \mid A \vee B, \Gamma \vdash (\forall x)A(x)} \vee \vdash'$$

where δ contains only structural inferences. We reduce the order of π by replacing this part of π by:

$$\dfrac{\dfrac{\dfrac{\vdots}{G \mid A, \Gamma \vdash \Delta} \quad \dfrac{\dfrac{\vdots\ \delta'}{G' \mid \Gamma' \vdash A(a)}}{\vdots\ \delta}}{G \mid A \vee B, \Gamma \vdash \Delta \mid A \vee B, \Gamma \vdash A(a)} \vee \vdash'}{G \mid A \vee B, \Gamma \vdash \Delta \mid A \vee B, \Gamma \vdash (\forall x)A(x)} \vdash \forall$$

□

5 Elimination of the Takeuti-Titani Rule

The Takeuti-Titani rule is the least understood feature of the original Takeuti-Titani axiomatization of **IF**. We show below that the rule can be eliminated from proofs in **HIF**. This had been posed as a problem by Takano [11]. The proof is by induction on the number of applications of (*tt*) and the length of the proof. The exact complexity of the elimination procedure is still to be investigated. The (*tt*) rule can have significant effects on proof structure. For instance, one of the calculi in Avron [2] uses the split rule

$$\frac{G \mid \Gamma, \Gamma' \vdash \Delta}{G \mid \Gamma \vdash \Delta \mid \Gamma' \vdash \Delta}\ \textit{split}$$

If this rule is added to **HIF**, it is possible to transform proofs so that each application of the communication rule has a premise which is a propositional axiom. This is not possible without (*tt*). The transformation works by replacing each occurrence of the communication rule by

$$\dfrac{\dfrac{q \vdash q \quad \dfrac{p \vdash p \quad \dfrac{G_1 \mid \Gamma_1, \Gamma_1' \vdash A_1}{G_1 \mid \Gamma_1 \vdash A_1 \mid \Gamma_1' \vdash A_1}\ \textit{split}}{G_1 \mid \Gamma_1 \vdash A_1 \mid \Gamma_1' \vdash p \mid p \vdash A_1}\ \textit{cm}}{G_1 \mid \Gamma_1 \vdash A_1 \mid \Gamma_1' \vdash q \mid p \vdash A_1 \mid q \vdash p}\ \textit{cm} \qquad \dfrac{\dfrac{G_2 \mid \Gamma_2, \Gamma_2' \vdash A_2}{G_2 \mid \Gamma_2 \vdash A_2 \mid \Gamma_2' \vdash A_2}\ \textit{split} \quad q \vdash q}{G_2 \mid \Gamma_2 \vdash q \mid q \vdash A_2 \mid \Gamma_2' \vdash A_2}\ \textit{cm}}{G_1 \mid G_2 \mid \Gamma_1 \vdash A_1 \mid \Gamma_2 \vdash q \mid p \vdash A_1 \mid \Gamma_2 \vdash p \mid q \vdash A_2 \mid \Gamma_2' \vdash A_2}\ \textit{cut}$$

$$\dfrac{\dfrac{G_1 \mid G_2 \mid \Gamma_1 \vdash A_1 \mid \Gamma_2 \vdash q \mid p \vdash A_1 \mid \Gamma_2 \vdash p \mid q \vdash A_2 \mid \Gamma_2' \vdash A_2}{G_1 \mid G_2 \mid \Gamma_1 \vdash A_1 \mid \Gamma_2 \vdash A_2 \mid p \vdash A_1 \mid \Gamma_2 \vdash p \mid \Gamma_2' \vdash A_2}\ \textit{tt}}{\dfrac{G_1 \mid G_2 \mid \Gamma_1 \vdash A_1 \mid \Gamma_2 \vdash A_2 \mid \Gamma_2 \vdash A_1 \mid \Gamma_2' \vdash A_2}{G_1 \mid G_2 \mid \Gamma_1, \Gamma_2' \vdash A_1 \mid \Gamma_1', \Gamma_2 \vdash A_2}}\ \textit{tt}$$

Proposition 8. *Let δ be a* **HIF****-derivation of hypersequent H with length k, where H is of the form*

$$G \mid \Gamma_1, \Pi_1 \vdash \Delta_1, \Pi_1' \mid \ldots \mid \Gamma_n, \Pi_n \vdash \Delta_n, \Pi_n'$$

and $\bigcup \Pi_i \subseteq \{p\}$, $\Pi_i' = \emptyset$, and p does not occur in G, Γ_i or Δ_i ($\bigcup \Pi_i' = \{p\}$, $\Pi_i = \emptyset$, and p does not occur in G, Γ_i or Δ_i).

Then the hypersequent $G \mid \Gamma_{i_1} \vdash \Delta_{i_1} \mid \ldots \mid \Gamma_{i_m} \vdash \Delta_{i_m}$ is derivable in length $\leq k$.

Proof. Easy induction on k. Every occurrence of p must arise from a weakening, simply delete all these weakenings.

Theorem 9. *Applications of (tt) can be eliminated from* **HIF***-derivations.*

This follows from the following lemma by induction on the number of applications of (*tt*) in a given **HIF**$^-$-derivation.

Lemma 10. *If δ is an* **HIF****-derivation of*

$$H = G \mid \Phi_1 \vdash \Pi_1 \mid \dots \Phi_n \vdash \Pi_n \mid \Pi'_1, \Psi_1 \vdash \Sigma_1 \mid \dots \mid \Pi'_m, \Psi_m \vdash \Sigma_m,$$

where p does not occur in G, Φ_i, Ψ_i or Σ_i, and $\bigcup \Pi_i \cup \bigcup \Pi'_i \subseteq \{p\}$, then there is a **HIF****-derivation of*

$$H^* = G \mid \Phi_1, \dots, \Phi_n, \Psi_1 \vdash \Sigma_1 \mid \dots \mid \Phi_1, \dots, \Phi_n, \Psi_m \vdash \Sigma_m.$$

Proof. By induction on the length of δ. We distinguish cases according to the last inference I in δ. For simplicity, we will write p in what follows below instead of Π_i or Π'_i with the understanding that it denotes an arbitrary multiset of p's.

(1) The conclusion of of I is so that p only occurs on the right side of sequents, or only on the left side. Then Prop. 8 applies, and the desired hypersequent can be derived without (tt).

(2) I applies to sequents in G. Then the induction hypothesis can be applied to the premise(s) of I and appropriate inferences added below.

(3) I is structural inference other than (cut) and (cm), or a logical inference with only one premise, or a logical inference which applies to a Σ_i. These cases are likewise handled in an obvious manner and are unproblematic. One instructive example might be the case of $(\supset\vdash)$. Here the premises would be of the form, say,

$$G \mid \Phi_1 \vdash p \mid \Phi_2 \vdash p \dots \mid \Phi_n \vdash p \mid p, \Psi_1 \vdash \Sigma_1 \mid \dots \mid p, \Psi_m \vdash \Sigma_m \mid p, \Gamma_1 \vdash A$$
$$G \mid \Phi_1 \vdash p \mid \Phi_2 \vdash p \dots \mid \Phi_n \vdash p \mid p, \Psi_1 \vdash \Sigma_1 \mid \dots \mid p, \Psi_m \vdash \Sigma_m \mid B, \Gamma_2 \vdash p$$

Let $\Phi = \Phi_1, \dots, \Phi_n$. The induction hypothesis provides us with

$$G \mid \Phi, \Psi_1 \vdash \Sigma_1 \mid \dots \mid \Phi, \Psi_m \vdash \Sigma_m \mid \Phi, \Gamma_1 \vdash A$$
$$G \mid B, \Gamma_2, \Phi, \Psi_1 \vdash \Sigma_1 \mid \dots \mid B, \Gamma_2, \Phi, \Psi_m \vdash \Sigma_m$$

We obtain the desired hypersequent by applying $(\supset\vdash)$ successively m times, together with some contractions.

(4) I is a cut. There are several cases to consider, most of which are routine. The only tricky case is when the cut formula is p and p occurs both on the left and the right side of sequents in both premises of the cut. For simplicity, let us consider the cut rule in its multiplicate formulation

$$G \mid \Phi_1 \vdash p \mid \dots \mid \Phi_n \vdash p \mid p, \Psi_1 \vdash \Sigma_1 \mid \dots \mid p, \Psi_m \vdash \Sigma_m \mid \Gamma \vdash p$$
$$G \mid \Phi_1 \vdash p \mid \dots \mid \Phi_n \vdash p \mid p, \Psi_1 \vdash \Sigma_1 \mid \dots \mid p, \Psi_m \vdash \Sigma_m \mid p, \Pi \vdash \Lambda$$

We want to find a derivation of

$$G \mid \Phi, \Psi_1 \vdash \Sigma_1 \mid \Phi, \Psi_m \vdash \Sigma_m \mid \Gamma, \Pi \vdash \Lambda$$

where $\Phi = \Phi_1, \dots, \Phi_n$. The induction hypothesis applied to the premises of the cut gives us

$$G \mid \Gamma, \Phi, \Psi_1 \vdash \Sigma_1 \mid \dots \mid \Gamma, \Phi, \Psi_m \vdash \Sigma_m$$
$$G \mid \Phi, \Psi_1 \vdash \Sigma_1 \mid \dots \mid \Phi, \Psi_m \vdash \Sigma_m \mid \Phi, \Pi \vdash \Lambda$$

We obtain the desired hypersequent by m successive applications of (cm).

(5) I is $(\vee\vdash)$, or $(\exists\vdash)$ applying to Φ_i or Ψ_i. Consider the case of $(\vee\vdash)$, the others are treated similarly. The premises of I are, for example,

$$G \mid A, \Phi_1 \vdash p \mid \Phi_2 \vdash p \ldots \mid \Phi_n \vdash p \mid p, \Psi_1 \vdash \Sigma_1 \mid \ldots \mid p, \Psi_m \vdash \Sigma_m$$
$$G \mid B, \Phi_1 \vdash p \mid \Phi_2 \vdash p \ldots \mid \Phi_n \vdash p \mid p, \Psi_1 \vdash \Sigma_1 \mid \ldots \mid p, \Psi_m \vdash \Sigma_m$$

By induction hypothesis, we obtain

$$G \mid A, \Phi_1, \ldots, \Phi_n, \Psi_1 \vdash \Sigma_1 \mid \ldots \mid A, \Phi_1, \ldots, \Phi_n, \Psi_m \vdash \Sigma_m$$
$$G \mid B, \Phi_1, \ldots, \Phi_n, \Psi_1 \vdash \Sigma_1 \mid \ldots \mid B, \Phi_1, \ldots, \Phi_n, \Psi_m \vdash \Sigma_m$$

It is not straightforwardly possible to derive the desired hypersequent from these. If $\Psi_i = \{P_{i1}, \ldots, P_{ik_i}\}$, let $Q_i = P_{i1} \supset \ldots P_{ik_i} \supset \Sigma_i$. Then we do easily obtain, however, the following by repeated application of $(\vdash\supset)$, $(\vdash\vee)$ and (ec):

$$G \mid A, \Phi_1, \ldots, \Phi_n \vdash Q_1 \vee \ldots \vee Q_m$$
$$G \mid B, \Phi_1, \ldots, \Phi_n \vdash Q_1 \vee \ldots \vee Q_m$$

Now a single application of $(\vee\vdash)$, plus (ec) gives us

$$K = G \mid \underbrace{A \vee B, \Phi_1, \ldots, \Phi_n}_{\Gamma} \vdash Q_1 \vee \ldots \vee Q_m$$

Then we derive, using $m-1$ cuts:

$$\dfrac{K \quad \dfrac{\vdots\ \delta_1}{Q_1 \vee Q \vdash Q_1 \mid Q_1 \vee Q \vdash Q}}{\Gamma \vdash Q_1 \mid \Gamma \vdash \underbrace{Q_2 \vee \ldots \vee Q_m}_{Q}} \qquad \dfrac{\Gamma \vdash Q_1 \mid \ldots \mid \Gamma \vdash Q_{m-1} \vee Q_m \quad Q_{m-1} \vee Q_m \vdash Q_{m-1} \mid Q_{m-1} \vee Q_m \vdash Q_m}{\Gamma \vdash Q_1 \mid \ldots \mid \Gamma \vdash Q_m}$$

where δ_i is the derivation

$$\dfrac{Q_i \vdash Q_i \qquad \dfrac{\dfrac{Q \vdash Q \quad Q_i \vdash Q_i}{Q \vdash Q_i \mid Q_i \vdash Q}\, cm \qquad Q \vdash Q}{Q \vdash Q_i \mid Q_i \vee Q \vdash Q}\, \vee\vdash}{Q_i \vee \underbrace{Q_{i+1} \vee \ldots \vee Q_m}_{Q} \vdash Q_i \mid Q_i \vee \ldots \vee Q_m \vdash \underbrace{Q_{i+1} \vee \ldots \vee Q_m}_{Q}}\, \vee\vdash$$

The desired hypersequent is obtained by m cuts with

$$Q_i, P_{i1}, \ldots, P_{ik_i} \vdash \Sigma_i$$

(6) I is a communication rule. This is the most involved case, as several subcases have to be distinguished according to which of the two communicated sequents contains p. Neither of these cases are problematic. We present two examples:

(a) One of the communicated sequents contains p on the right. Then the premises of I are

$$G \mid \Phi_1 \vdash p \mid \ldots \mid \Phi_n \vdash p \mid p, \Psi_1 \vdash \Sigma_1 \mid \ldots \mid p, \Psi_m \vdash \Sigma_m \mid \Theta_1, \Theta_1' \vdash p$$
$$G \mid \Phi_1 \vdash p \mid \ldots \mid \Phi_n \vdash p \mid p, \Psi_1 \vdash \Sigma_1 \mid \ldots \mid p, \Psi_m \vdash \Sigma_m \mid \Theta_2, \Theta_2' \vdash \Xi_2$$

where. The induction hypothesis applies to these two hypersequents. If we write $\Phi = \Phi_1, \ldots, \Phi_n$, we have

$$G \mid \boxed{\Theta_1, \Theta_1', \Phi, \Psi_1 \vdash \Sigma_1} \mid \ldots \mid \Theta_1, \Theta_1', \Phi, \Psi_m \vdash \Sigma_m$$
$$G \mid \boxed{\Theta_2, \Theta_2' \vdash \Xi} \mid \Phi, \Psi_1 \vdash \Sigma_1 \mid \ldots \mid \Phi, \Psi_m \vdash \Sigma_m$$

We obtain the desired result by applying m instances of (cm), internal weakenings and external contractions as necessary, to obtain, in sequence

$$G \mid \Theta_1, \Theta_2', \Phi, \Psi_1 \vdash \Sigma_1 \mid \ldots \mid \Theta_1, \Theta_1', \Phi, \Psi_m \vdash \Sigma_m \mid \Theta_1', \Theta_2 \vdash \Xi$$
$$\vdots$$
$$G \mid \Theta_1, \Theta_2', \Phi, \Psi_1 \vdash \Sigma_1 \mid \ldots \mid \Theta_1, \Theta_2', \Phi, \Psi_m \vdash \Sigma_m \mid \Theta_1', \Theta_2 \vdash \Xi$$

The sequents participating in the application of (cm) are marked by boxes. The original end hypersequent follows from the last one by internal weakenings.

(b) The communicated sequents both contain p, once on the right, once on the left. The premises of I are

$$G \mid \Phi_1 \vdash p \mid \ldots \mid \Phi_n \vdash p \mid p, \Psi_1 \vdash \Sigma_1 \mid \ldots \mid p, \Psi_m \vdash \Sigma_m \mid \Theta_1, \Theta_1' \vdash p$$
$$G \mid \Phi_1 \vdash p \mid \ldots \mid \Phi_n \vdash p \mid p, \Psi_1 \vdash \Sigma_1 \mid \ldots \mid p, \Psi_m \vdash \Sigma_m \mid p, \Theta_2, \Theta_2' \vdash \Xi$$

We have proofs of

$$G \mid \Theta_1, \Theta_1', \Phi, \Psi_1 \vdash \Sigma_1 \mid \ldots \mid \Theta_1, \Theta_1', \Phi, \Psi_m \vdash \Sigma_m$$
$$G \mid \Phi, \Psi_1 \vdash \Sigma_1 \mid \ldots \mid \Phi, \Psi_m \vdash \Sigma_m \mid \Theta_2, \Theta_2', \Phi \vdash \Xi$$

Again, a sequence of m applications of (cm), together with internal weakenings and external contractions produces the desired end sequent. □

Note that in case (5), several new cuts are introduced. As a consequence, the elimination procedure does not directly work for cut-free proofs. If a proof with neither cut nor communication is required, the elimination procedure has to be

combined with the cut-elimination procedure of Thm. 5. The additional cuts can be avoided by replacing $(\vee\vdash)$ and $(\exists\vdash)$ by the following generalized rules:

$$\frac{G \mid A, \Gamma_1 \vdash \Delta_1 \mid \ldots \mid A, \Gamma_n \vdash \Delta_n \quad G \mid B, \Gamma_1 \vdash \Delta_1 \mid \ldots \mid B, \Gamma_n \vdash \Delta_n}{G \mid A \vee B, \Gamma_1 \vdash \Delta_1 \mid \ldots \mid A \vee B, \Gamma_n \vdash \Delta_n} \vee\vdash^*$$

$$\frac{G \mid A(a), \Gamma_1 \vdash \Delta_1 \mid \ldots \mid A(a), \Gamma_n \vdash \Delta_n}{G \mid (\exists x)A(x), \Gamma_1 \vdash \Delta_1 \mid \ldots \mid (\exists x)A(x), \Gamma_n \vdash \Delta_n} \exists\vdash^*$$

These rules, however, cannot be simulated by the ordinary rules without using cut (the simulation with cut is given in case (5)). By changing case (5) accordingly, the elimination procedure will transform a cut-free **HIF**-derivation into a cut-free one without (tt), but with $(\vee\vdash^*)$ and $(\exists\vdash^*)$.

References

[1] A. Avellone, M. Ferrari, P. Miglioli, and U. Moscato. A tableau calculus for Dummett predicate logic. In W. A. Carnielli and I. M. L. D'Ottaviano, editors, *Advances in Contemporary Logic and Computer Science*, Contemporary Mathematics 235, 135–151. American Mathematical Society, Providence, 1999.

[2] A. Avron. Hypersequents, logical consequence and intermediate logics for concurrency. *Ann. Math. Artificial Intelligence*, 4:225–248, 1991.

[3] M. Baaz and H. Veith. An axiomatization of quantified propositional Gödel logic using the Takeuti-Titani rule. In S. Buss, P. Hájek, and P. Pudlák, editors, *Logic Colloquium '98. Proceedings*, LNL 13, 74–87. ASL, 2000.

[4] G. Corsi. A cut-free calculus for Dummett's LC quantified. *Z. Math. Logik Grundlag. Math.*, 35:289–301, 1989.

[5] G. Corsi. Completeness theorem for Dummett's LC quantified and some of its extensions. *Studia Logica*, 51:317–335, 1992.

[6] M. Dummett. A propositional calculus with denumerable matrix. *J. Symbolic Logic*, 24:97–106, 1959.

[7] D. M. Gabbay. Decidability of some intuitionistic predicate theories. *J. Symbolic Logic*, 37:579–587, 1972.

[8] K. Gödel. Zum intuitionistischen Aussagenkalkül. *Anz. Akad. Wiss. Wien*, 69:65–66, 1932.

[9] P. Hájek. *Metamathematics of Fuzzy Logic*. Kluwer, Dordrecht, 1998.

[10] A. Horn. Logic with truth values in a linearly ordered Heyting algebra. *J. Symbolic Logic*, 34:395–408, 1969.

[11] M. Takano. Another proof of the strong completeness of the intuitionistic fuzzy logic. *Tsukuba J. Math*, 11:101–105, 1987.

[12] G. Takeuti. *Proof Theory*. North-Holland, Amsterdam, 2nd ed., 1987.

[13] G. Takeuti and S. Titani. Intuitionistic fuzzy logic and intuitionistic fuzzy set theory. *J. Symbolic Logic*, 49:851–866, 1984.

Continuous Functionals of Dependent Types and Equilogical Spaces

Andrej Bauer[1] and Lars Birkedal[2]

[1] School of Computer Science, Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh PA 15213, USA
Andrej.Bauer@cs.cmu.edu
[2] The IT University of Copenhagen
Glentevej 67, DK-2400 København NV, Denmark
birkedal@it-c.dk

Abstract. We show that dependent sums and dependent products of continuous parametrizations on domains with dense, codense, and natural totalities agree with dependent sums and dependent products in equilogical spaces, and thus also in the realizability topos $\mathsf{RT}(\mathcal{P}\omega)$.

Keywords: continuous functionals, dependent type theory, domain theory, equilogical spaces.

1 Introduction

Recently there has been a lot of interest in understanding notions of totality for domains [3,23,4,18,21]. There are several reasons for this. Totality is the semantic analogue of termination, and one is naturally interested in understanding not only termination properties of programs but also how notions of program equivalence depend on assumptions regarding termination [21]. Another reason for studying totality on domains is to obtain generalizations of the finite-type hierarchy of total continuous functionals by Kleene and Kreisel [11], see [8] and [19] for good accounts of this subject. Ershov [7] showed how the Kleene-Kreisel functionals arise in a domain-theoretic setting as the *total* elements of domains of *partial* continuous functionals. This work has been pursued further by Normann, Berger and others, who have studied both inductive types and dependent types with universe operators [3,23,4,18,12,26]. The aims of their work include both finding models of Martin-Löf type theory [16,26] and also extending the *density theorems* to transfinite hierarchies. The density theorems are used in the study of higher-type recursion theory and in order-theoretic characterizations of extensionality for total objects [4,17].

It is important to understand how different models of computation relate. Indeed, a number of results demonstrate that the Kleene-Kreisel functionals arise in various computational models [7,10,15,3,13], which is good evidence that this class of functionals is an important and robust model of higher-type computation. We proved one such result in [2], where we related domains with totality to *equilogical spaces*, introduced by Dana Scott [2]: the so-called dense and codense totalities on domains [3] embed fully and faithfully into the category of

P. Clote and H. Schwichtenberg (Eds.): CSL 2000, LNCS 1862, pp. 202–216, 2000.

equilogical spaces and the embedding preserves the cartesian-closed structure implicit in the totalities for products and function spaces. From this it follows easily that the Kleene-Kreisel functionals of finite type can be constructed in the category Equ of equilogical spaces by repeated exponentiation, starting from the natural numbers object. In this paper we extend these results to dependent types.

We build on Berger's Habilitationsschrift [4], in which Berger generalized density and codensity on domains from simple types to dependent types with universe operators and proved the corresponding Density Theorems. We show that, in a precise sense, the dependent types of dense, codense, and natural totalities on consistent parametrizations coincide with the dependent types of equilogical spaces. It follows that the dependent type hierarchies over the natural numbers and the booleans coincide in four settings: equilogical spaces, domains with totality, limit spaces [20], and filter spaces [22,9]. We also note recent work by Menni and Simpson [14], which relate locally cartesian closed subcategories of equilogical spaces, sequential spaces, and limit spaces. All these results taken together provide a satisfactory "goodness of fit" picture, at the level of *dependent* type structures.

More precisely, domains here are algebraic, countably based, consistently-complete dcpos. Since the domains are countably based, we only need to consider countably based equilogical spaces, which form a full locally cartesian closed subcategory of the category of all equilogical spaces. The category of countably based equilogical spaces is equivalent to the category of modest sets $\mathsf{Mod}(\mathcal{P}\omega)$ over the graph model $\mathcal{P}\omega$ of the untyped λ-calculus, and since the modest sets form a full locally cartesian closed subcategory of the realizability topos $\mathsf{RT}(\mathcal{P}\omega)$ over the graph model, it follows that the domain-theoretic total continuous functionals of dependent types are the same as the ones in the realizability topos $\mathsf{RT}(\mathcal{P}\omega)$.

The plan of the paper is as follows. In the following section we present an overview of the technical work, and explain the main idea of the proof of our main theorem, Theorem 1. In Sect. 3 we recall the definition of the category of equilogical spaces and the construction of dependent sums and products of equilogical spaces. In Sect. 4 we briefly review domains with totality, and refer you to [4] for more details. Sect. 5 contains the Main Theorem and its proof, which relates dependent types in Equ to dependent types in domains with totality. As an example of how the Main Theorem can be used, we translate Berger's Continuous Choice Principle for dependent totalities [4] into a choice principle expressed in the internal logic of Equ. Finally, Sect. 7 contains some concluding remarks and suggestions for future work.

Acknowledgments

This work is part of our Ph.D. research, carried out under the guidance of Dana Scott, whom we thank. We are also grateful to Ulrich Berger, Dag Normann, Pino Rosolini, and Alex Simpson for stimulating discussions about the subject.

2 Overview of Technical Work

In this section we give a brief overview of the rather technical theorems and proofs from Sect. 5. We do not provide any proofs or references for the claims made in this overview, because they are repeated in more detail in the rest of the paper. Please consult Sects. 3 and 4 for basic definitions and explanation of the notation. Berger [4,5] contains material on totalities for parametrizations on domains, and [2] can serve as a reference on equilogical spaces.

The category of countably based equilogical spaces, as defined originally by Dana Scott, is equivalent to $\mathsf{PER}(\omega\mathsf{ALat})$, the category of partial equivalence relations on countably based algebraic lattices. We work exclusively with $\mathsf{PER}(\omega\mathsf{ALat})$ and so we set $\mathsf{Equ} = \mathsf{PER}(\omega\mathsf{ALat})$.

If $M \subseteq D$ is a codense subset of a domain D, then the consistency relation $\uparrow$ (which relates two elements when they are bounded) restricted to M is a partial equivalence relation on D. Thus, a codense subset of a domain D can be viewed as a partial equivalence relation, induced by the consistency relation on M, on the algebraic lattice $D^\top$, the domain D with a compact top element $\top$ added to it.

Let $F = (|F|, \|F\|)$ be a dense, codense and consistent totality on $D = (|D|, \|D\|)$, i.e., $(|F|, |D|)$ is a consistent parametrization on the domain $|D|$, $\|D\| \subseteq |D|$ is a dense and codense totality on $|D|$, and $(\|D\|, \|F\|)$ is a dense and codense dependent totality for $|F|$. We can explain the main point of the proof that the dependent types in domains with totality agree with dependent types in equilogical spaces by looking at how the dependent products are constructed in both setting. In the domain-theoretic setting a total element of the dependent product $P = \Pi(D, F)$ is a continuous map $f = \langle f_1, f_2 \rangle \colon |D| \to |\Sigma(D, F)|$ that maps total elements to total elements and satisfies, for all $x \in \|D\|$, $f_1 x = x$. In $\mathsf{PER}(\omega\mathsf{ALat})$ a total element of the dependent product $Q = \prod_D F$ is a continuous map $g = \langle g_1, g_2 \rangle \colon |D|^\top \to |\Sigma(D, F)|^\top$ that preserves the partial equivalence relations and satisfies, for all $x \in \|D\|$, $g_1 x \uparrow_D x$. Here $\uparrow_D$ is the consistency relation on domain $|D|$, restricted to the totality $\|D\|$. In order to prove that P and Q are isomorphic we need to be able to translate an element $f \in \|P\|$ to one in $\|Q\|$, and vice versa. It is easy enough to translate $f \in \|P\|$ since we can just use f itself again. This is so because $f_1 x = x$ implies $f_1 x \uparrow_D x$. However, given a $g \in \|Q\|$, it is not obvious how to get a corresponding function in $\|P\|$. We need a way of continuously *transporting* 'level' $\|F(g_1 x)\|$ to 'level' $\|Fx\|$. In other words, we need a continuous map t such that whenever $x, y \in \|D\|$, $x \uparrow y$, and $u \in \|Fy\|$ then $t(y, x)u \in \|Fx\|$ and $\langle x, t(y, x)u \rangle \uparrow \langle y, u \rangle$ in $|\Sigma(D, F)|$. Given such a map t, the element of $\|P\|$ corresponding to $g \in \|Q\|$ is the map

$$x \mapsto \langle x, t(g_1 x, x)(g_2 x) \rangle.$$

The theory of totality for parametrizations on domains provides exactly what we need. Every consistent parametrization F has a *transporter* t, which has the desired properties. In addition, we must also require that the parametrization F be *natural*, which guarantees that $t(y, x)$ maps $\|Fy\|$ to $\|Fx\|$ whenever x and y

are total and consistent. Berger [4] used the naturality conditions for dependent totalities to show that the consistency relation coincides with extensional equality. As equality of functions in equilogical spaces is defined extensionally, it is not surprising that naturality is needed in order to show the correspondence between the equilogical and domain-theoretic settings.

Finally, let us comment on the significance of the density and codensity theorems [4] for the results presented in this paper. We define a translation from dependent totalities to equilogical spaces, and show that it preserves dependent sums and products. The density theorems for dependent totalities ensure that the translation is well defined in the first place. Thus, density plays a fundamental role, which is further supported by the observation that the category of equilogical spaces is equivalent to the category of *dense* partial equivalence relations on Scott domains, see [2]. The effect of codensity is that the translation of domain-theoretic totalities into equilogical spaces gives a rather special kind of *totally disconnected equilogical spaces*, which we comment on further in Sect. 7.

3 Equilogical Spaces

In this paper, we take an *equilogical space* $A = (|A|, \approx_A)$ to be a partial equivalence relation $\approx_A$ on an algebraic lattice $|A|$. The category $\mathsf{PER}(\omega\mathsf{ALat})$ of such objects and equivalence classes of equivalence preserving continuous maps between them is equivalent to the original definition of equilogical spaces [2].

The *support* of an equilogical space A is the set

$$\|A\| = \left\{ x \in |A| \;\middle|\; x \approx_A x \right\}.$$

We explicitly describe the locally cartesian closed structure of $\mathsf{PER}(\omega\mathsf{ALat})$.

Let $r\colon J \to I$ be a morphism in $\mathsf{PER}(\omega\mathsf{ALat})$. The *pullback along* r^* is the functor

$$r^*\colon \mathsf{PER}(\omega\mathsf{ALat})/I \to \mathsf{PER}(\omega\mathsf{ALat})/J$$

that maps an object $a\colon A \to I$ over I to an object $r^*a\colon r^*A \to J$ over J, as in the pullback diagram

$$\begin{array}{ccc} r^*A & \longrightarrow & A \\ {\scriptstyle r^*a}\downarrow & \lrcorner & \downarrow{\scriptstyle a} \\ J & \xrightarrow[r]{} & I \end{array}$$

The pullback functor r^* has left and right adjoints. The left adjoint is the *dependent sum along* r

$$\textstyle\sum_r\colon \mathsf{PER}(\omega\mathsf{ALat})/J \to \mathsf{PER}(\omega\mathsf{ALat})/I$$

that maps an object $b\colon B \to J$ over J to the the object $\sum_r b = r \circ b\colon B \to I$ over I. The right adjoint to the pullback functor r^* is the *dependent product along* r

$$\prod_r \colon \mathsf{PER}(\omega\mathsf{ALat})/J \to \mathsf{PER}(\omega\mathsf{ALat})/I,$$

defined as follows. Let $b\colon B \to J$ be an object in the slice over J. Let $\sim$ be a partial equivalence relation on the algebraic lattice $|I| \times (|J| \to |B|)$ defined by

$$\begin{gathered}\langle i, f\rangle \sim \langle i', f'\rangle \\ \text{if and only if} \\ i \approx_I i' \wedge \forall\, j, j' \in |J|\,.\,(j \approx_J j' \wedge r(j) \approx_I i \Longrightarrow f(j) \approx_B f'(j') \wedge b(f(j)) \approx_J j)\end{gathered}$$

The dependent product $\prod_r b$ is the object $(|\prod_r b|, \sim)$, where

$$|\textstyle\prod_r b| = |I| \times (|J| \to |B|)\ . \tag{1}$$

The map $\prod_r b\colon \prod_r b \to I$ is the obvious projection $\langle i, f\rangle \mapsto i$. See [2] for more details about the locally cartesian closed structure of $\mathsf{PER}(\omega\mathsf{ALat})$.

For background material on domain theory we suggest [24] or [1]. A *Scott domain* is a countably based, algebraic, consistently-complete dcpo. Let $\omega\mathsf{Dom}$ be the category of Scott domains and continuous maps between them. This category is cartesian closed and contains the category $\omega\mathsf{ALat}$ as a full cartesian closed subcategory. We define the 'top' functor $\square^\top : \omega\mathsf{Dom} \to \omega\mathsf{ALat}$ by setting $D^\top$ to be the domain D with a new compact top element added to it. Given a map $f\colon D \to E$, let $f^\top\colon D^\top \to E^\top$ be defined by

$$f^\top x = \begin{cases} fx & \text{if } x \neq \top_D \\ \top_E & \text{if } x = \top_D \end{cases}\ .$$

It is is easily checked that $f^\top$ is a continuous map. We are going to use the following two lemmas and corollary later on. The easy proofs are omitted.

Lemma 1. *Let C, D, and E be Scott domains and $f\colon C \to (D \to E^\top)$ a continuous map. Then the map $f'\colon C \to (D^\top \to E)$ defined by*

$$f'xy = \begin{cases} fxy & \text{if } y \neq \top_D \\ \top_E & \text{if } y = \top_D \end{cases}$$

is also continuous.

Corollary 1. *Let D, and E be Scott domains and $f\colon D \to E^\top$ a continuous map. Then the map $f'\colon D^\top \to E$ defined by*

$$f'y = \begin{cases} fy & \text{if } y \neq \top_D \\ \top_E & \text{if } y = \top_D \end{cases}$$

is also continuous.

Lemma 2. *Suppose D and E are Scott domains, $S \subseteq D$ is an open subset, and $f\colon D \setminus S \to E^\top$ is a continuous map from the Scott domain $D \setminus S$ to the algebraic lattice $E^\top$. Then the map $f'\colon D \to E^\top$ defined by*

$$f'x = \begin{cases} fx & \text{if } x \notin S \\ \top_E & \text{if } x \in S \end{cases}$$

is also continuous.

4 Domains and Totality

We review some basic definitions about domains with totality from Berger [3,4]. Let $\mathbb{B}_\perp$ be the flat domain on the Booleans $\mathbb{B} = \{\mathsf{false}, \mathsf{true}\}$. Given a domain D and a subset $M \subseteq D$, let $\mathcal{E}_D(M)$ be the family

$$\mathcal{E}_D(M) = \left\{ p\colon D \to \mathbb{B}_\perp \;\middle|\; \forall\, x \in M\,.\, px \neq \perp \right\}.$$

In words, $\mathcal{E}_D(M)$ is the set of those continuous predicates on D which only take on values true and false on elements of M. The family $\mathcal{E}_D(M)$ is *separating* when for every unbounded finite set $\{x_0, \ldots, x_n\} \subseteq D$, there exist $p_0, \ldots, p_n \in \mathcal{E}_D(M)$ such that $p_i x_i = \mathsf{true}$ for $i = 0, \ldots, n$ and $p_0^*(\mathsf{true}) \cap \cdots \cap p_n^*(\mathsf{true}) = \emptyset$.

A *totality* on a domain is a pair $D = (|D|, \|D\|)$ where $|D|$ is a domain and $\|D\|$ is a subset of $|D|$. Often the set $\|D\|$ itself is called a totality as well. A totality is *dense* when $\|D\|$ is a topologically dense subset of $|D|$. A totality is *codense* when the family $\mathcal{E}_{|D|}(\|D\|)$ is separating. The consistency relation $\uparrow$ restricted to a codense totality $\|D\|$ is symmetric and transitive.

To each dense and codense totality D we assign an equilogical space

$$\mathrm{Q}D = (|D|^\top, \uparrow_D) \tag{2}$$

where $\uparrow_D$ is the consistency relation restricted to the totality $\|D\|$, i.e., $x \uparrow_D y$ if, and only if, $x, y \in \|D\| \wedge x \uparrow y$. We consider only dense and codense totalities from now on.

A *parametrization* on a domain $|D|$ is a co-continuous functor $F\colon |D| \to \omega\mathsf{Dom}^{\mathrm{ep}}$ from $|D|$, viewed as a category, to the category $\omega\mathsf{Dom}^{\mathrm{ep}}$ of Scott domains and *good* embeddings. Recall from [4] that an embedding-projection pair is good when the projection preserves arbitrary suprema. Whenever $x, y \in |D|$, $x \leq y$, there is an embedding $F(x \leq y)^+\colon Fx \to Fy$ and a projection $F(x \leq y)^-\colon Fy \to Fx$. We abbreviate these as follows, for $u \in Fx$ and $v \in Fy$:

$$u^{[y]} = F(x \leq y)^+(u)\ ,$$
$$v_{[x]} = F(x \leq y)^-(v)\ .$$

A parametrization F on $|D|$ is *consistent* when it has a *transporter*. A transporter is a continuous map t such that for every $x, y \in |D|$, $t(x, y)$ is a map from Fx to Fy, satisfying:

(1) if $x \leq y$ then $F(x \leq y)^+ \leq t(x,y)$ and $F(x \leq y)^- \leq t(y,x)$,
(2) $t(x,y)$ is strict,
(3) $t(y,z) \circ t(x,y) \leq t(x,z)$.

Let D be a totality. A *dependent totality* on D is a pair $F = (|F|, \|F\|)$ where $|F| \colon |D| \to \omega\mathsf{Dom}^{\mathrm{ep}}$ is a parametrization and $(\|D\|, \|F\|)$ is a totality for the parametrization $(|D|, |F|)$. Just like for totalities on domains, there are notions of *dense* and *codense* dependent totalities. See Berger [4] for definitions of these and also for definitions of *dependent sum* $\Sigma(D,F)$ and *dependent product* $\Pi(D,F)$. From now on we only consider dense and codense dependent totalities on consistent parametrizations.

A dependent totality F on D is *natural* if $\|D\|$ is upward closed in $|D|$, $\|Fx\|$ is upward closed in $|Fx|$ for all $x \in \|D\|$, and whenever $x \leq y \in \|D\|$ then

$$\forall\, v \in |Fy| \,.\, \big(v \in \|Fy\| \iff v_{[x]} \in \|Fx\|\big) \quad .$$

Note that the above condition implies

$$\forall\, u \in |Fx| \,.\, \Big(u \in \|Fx\| \iff u^{[y]} \in \|Fy\|\Big) \quad .$$

Lemma 3. *Let F be a natural dependent totality on D. Since F is consistent, it has a transporter t. Let $x, y \in \|D\|$, $x \uparrow y$, and $u \in \|Fy\|$. Then $t(y,x)u \in \|Fx\|$ and $\langle y, u\rangle \uparrow \langle x, t(y,x)u\rangle$ in $|\Sigma(D,F)|$.*

Proof. By naturality of F we have $(u^{[x\vee y]})_{[x]} \in \|Fx\|$, and since

$$(u^{[x\vee y]})_{[x]} \leq t(x \vee y, x)(t(y, x \vee y)u) \leq t(y,x)u$$

also $t(y,x)u \in \|Fx\|$. Furthermore, $\langle y, u\rangle \uparrow \langle x, t(y,x)u\rangle$ in $|\Sigma(D,F)|$ because $x \uparrow y$ and $u^{[x\vee y]} \uparrow (t(y,x)u)^{[x\vee y]}$, which follows from the common upper bound

$$\begin{aligned} u^{[x\vee y]} &\leq t(y, x \vee y)u, \\ (t(y,x)u)^{[x\vee y]} &\leq (t(x, x \vee y) \circ t(y,x))u \leq t(y, x \vee y)u \quad . \end{aligned}$$

This completes the proof.

Let F be a dependent totality on D and let G be a dependent totality on $\Sigma(D,F)$. Define a *parametrized* dependent totality $\widetilde{G}$, i.e., a co-continuous functor from D to the category of parametrizations [4], by

$$\widetilde{G}x = \lambda u \in Fx \,.\, G(x,u) \quad .$$

More precisely, for each $x \in D$, $\widetilde{G}x$ is a dependent totality on Fx, defined by the curried form of G as above. In [4], which provides more details, $\widetilde{G}$ is called the *large currying* of G. Given such a $\widetilde{G}$, there are parametrized versions of

dependent sum $\Sigma(F,G)$ and dependent product $\Pi(F,G)$, which are dependent totalities on D, defined for $x \in D$ by

$$\Pi(F,G)x = \Pi(Fx, \widetilde{G}x) \ ,$$
$$\Sigma(F,G)x = \Sigma(Fx, \widetilde{G}x) \ .$$

To each natural dependent totality F on D we assign an equilogical space

$$\mathrm{q}(D,F)\colon \mathrm{Q}(D,F) \to \mathrm{Q}D$$

in the slice over $\mathrm{Q}D$ by defining

$$\mathrm{Q}(D,F) = \mathrm{Q}(\Sigma(D,F)) \tag{3}$$
$$\mathrm{q}(D,F) = \pi_1^{\top} \ , \tag{4}$$

where π_1 is the first projection $\pi_1\colon |\Sigma(D,F)| \to |D|$, $\pi_1\colon \langle x,u\rangle \mapsto x$.

5 Comparison of Dependent Types

We show that dependent sums and products on totalities coincide with those on equilogical spaces.

Theorem 1 (Main Theorem). *Let F be a dependent totality on D, and let G be a dependent totality on $\Sigma(D,F)$. The construction of dependent sum $\Sigma(F,G)$ and dependent product $\Pi(F,G)$ agrees with the construction of dependent sum and dependent product in* $\mathsf{PER}(\omega\mathsf{ALat})$*, i.e.,*

$$\mathrm{Q}(D,\Sigma(F,G)) \cong \textstyle\sum_{\mathrm{q}(D,F)} \mathrm{q}(\Sigma(D,F),G) \ ,$$
$$\mathrm{Q}(D,\Pi(F,G)) \cong \textstyle\prod_{\mathrm{q}(D,F)} \mathrm{q}(\Sigma(D,F),G)$$

in the slice over $\mathrm{Q}D$.

The rest of this section constitutes a proof of the Main Theorem, but before we embark on it, let us explain its significance. We have defined a translation Q from domain-theoretic dependent totalities to equilogical spaces. The Main Theorem says that this translation commutes with the construction of dependent sums and products. Thus, Q preserves the implicit local cartesian closed structure of totalities $\Sigma(F,G)$ and $\Pi(F,G)$. It may seem odd that we did not define a *functor* Q that would embed the dependent totalities into $\mathsf{PER}(\omega\mathsf{ALat})$ and preserve the locally cartesian closed structure. This can be done easily enough, by defining the morphisms $(D,F) \to (E,G)$ to be (equivalence classes of) equivalence-preserving continuous maps $\mathrm{Q}(D,F) \to \mathrm{Q}(E,G)$, i.e., essentially as the morphisms in $\mathsf{PER}(\omega\mathsf{ALat})$. Note that this is different from the definition of morphisms between parametrizations, as defined in Berger [4], where the motivation was to build the hierarchies in the first place, rather than to study an interpretation of dependent type theory. Thus, a notion of morphism

suitable for the interpretation of dependent type theory was never explicitly given, although it is fairly obvious what it should be. In this manner we trivially obtain a full and faithful functor Q. The crux of the matter is that with such a choice of morphisms, the domain-theoretic constructions $\Sigma(F, G)$ and $\Pi(F, G)$ indeed yield the category-theoretic dependent sums and products. This is the main purpose of our work—to show that the domain theoretic constructions of dependent functionals, which has at times been judged arcane and ad hoc, is essentially the same as the dependent functionals arising in the realizability topos $\mathsf{RT}(\mathcal{P}\omega)$, which is much smoother and better understood from the category-theoretic point of view. The benefits of this correspondence go both ways. On the one hand, the domain-theoretic construction, which was conceived through a sharp conceptual analysis of the underlying domain-theoretic notions, is more easily understood and accepted by a category theorist. On the other hand, we can transfer the domain-theoretic results about the dependent functionals to Equ and $\mathsf{RT}(\mathcal{P}\omega)$, e.g., the Continuous Choice Principle from Sect 6. It is not clear how to obtain the Continuous Choice Principle directly in the realizability setting.

Lastly, we note that the Main Theorem is formulated for dependent sums and products with *parameters*, i.e., for parametrizations of parametrizations on domains; a parameter-free formulation states only that $Q(\Pi(D, F)) \cong \prod q(D, F)$. We need the theorem with parameters in order to establish the full correspondence between the lccc structures. We now proceed with the proof of the Main Theorem.

Dependent Sums. Dependent sums are easily dealt with because all we have to do is unravel all the definitions. For this purpose, let $X = \mathrm{Q}(D, \Sigma(F, G))$ and $Y = \sum_{\mathrm{q}(D,F)} \mathrm{q}(\Sigma(D, F), G)$. In order to simplify the presentation we assume that ordered pairs and tuples satisfy the identities $\langle x, y, z\rangle = \langle\langle x, y\rangle, z\rangle = \langle x, \langle y, z\rangle\rangle$. This does affect the correctness of the proof, since it just amounts to leaving out the appropriate canonical isomorphisms. In particular, this assumption implies the equality $|\Sigma(\Sigma(D, F), G)| = |\Sigma(D, \Sigma(F, G))|$. From this it follows that the underlying lattices $|X|$ and $|Y|$ agree because

$$|Y| = |\Sigma(\Sigma(D, F), G)|^{\top} = |\Sigma(D, \Sigma(F, G))|^{\top} = |X| \ .$$

It remains to show that the partial equivalence relations on X and Y agree as well. We omit the straightforward verification of this fact.

Dependent Products. Dependent products are more complicated. There seems to be no way around it, since we are dealing with rather heavy domain-theoretic machinery. Let

$$\begin{aligned} U &= \mathrm{Q}(D, \Pi(F, G)) \ , \\ V &= \textstyle\prod_{\mathrm{q}(D,F)} \mathrm{q}(\Sigma(D, F), G) \ . \end{aligned}$$

Let us explicitly describe U and V. The underlying lattice of U is

$$|U| = |\Sigma(D, \Pi(F, G))|^{\top} \ . \tag{5}$$

The partial equivalence relation on U relates $\langle x, f\rangle \in |U|$ and $\langle y, g\rangle \in |U|$ if, and only if,

$$\begin{aligned}
&x \uparrow_D y \ \wedge \\
&(\forall\, u \in \|Fx\| \,.\, fu \in \|G(x,u)\|) \wedge (\forall\, v \in \|Fy\| \,.\, gv \in \|G(y,v)\|) \ \wedge \\
&\forall\, w \in |F(x \vee y)| \,.\, \Big((f(w_{[x]}))^{[\langle x \vee y, w\rangle]} \uparrow (g(w_{[y]}))^{[\langle x \vee y, w\rangle]}\Big) \ .
\end{aligned}$$

By (1), the underlying lattice of V is

$$|V| = |D|^{\top} \times \big(|\Sigma(D,F)|^{\top} \to |\Sigma(\Sigma(D,F),G)|^{\top}\big) \ . \tag{6}$$

Elements $\langle x, y\rangle \in |V|$ and $\langle y, g\rangle \in |V|$ are related if, and only if, the following holds: $x \uparrow_D y$, and for all $z, z' \in |D|$ such that $z \uparrow_D x$ and $z' \uparrow_D x$, and for all $w \in |Fz|$, $w' \in |Fz'|$ such that $w^{[z \vee z']} \uparrow_{F(z \vee z')} w'^{[z \vee z']}$,

$$\begin{aligned}
&f\langle z, w\rangle \uparrow_{\Sigma(\Sigma(D,F),G)} g(z', w') \ \wedge \\
&\pi_1(f\langle z, w\rangle) \uparrow_{\Sigma(D,F)} \langle z, w\rangle \ \wedge \ \pi_1(g\langle z', w'\rangle) \uparrow_{\Sigma(D,F)} \langle z', w'\rangle \ .
\end{aligned}$$

We define maps $\phi\colon |U| \to |V|$ and $\theta\colon |V| \to |U|$, and verify that they represent isomorphisms between U and V. Let t be a transporter for the parametrization F. Define the map $\phi\colon |U| \to |V|$ by

$$\phi\top = \top \ , \qquad \phi(x,f) = \langle x, \phi_2(x,f)\rangle \ ,$$

where $\phi_2(x,f)\colon |\Sigma(D,F)|^{\top} \to |\Sigma(\Sigma(D,F),G)|^{\top}$ is

$$\phi_2(x,f)\top = \top \ , \qquad \phi_2(x,f)(y,u) = \langle x, t(y,x)u, f(t(y,x)u)\rangle \ .$$

Let s be a transporter for the parametrization G on $\Sigma(D,F)$. Define the map $\theta\colon |V| \to |U|$ by

$$\begin{aligned}
\theta(\top, g) &= \top \\
\theta(x, g) &= \texttt{if} \ \ \exists\, u \in |Fx| \,.\, g(x,u) = \top \\
&\quad\ \texttt{then} \ \ \top \\
&\quad\ \texttt{else} \ \ \langle x, \lambda u \in |Fx| \,.\, s(g_1(x,u), \langle x, u\rangle)(g_2(x,u))\rangle
\end{aligned}$$

where $g = \langle g_1, g_2\rangle\colon |\Sigma(D,F)| \to |\Sigma(\Sigma(D,F),G)|$.

It is easy and tedious to verify that ϕ and θ have the intended types. Continuity of ϕ follows directly from Corollary 1 and Lemma 1. Continuity of θ follows from Lemmas 1 and 2. We can apply Lemma 2 because the set

$$\big\{\langle x, g\rangle \;\big|\; \exists\, u \in |Fx| \,.\, g(x,u) = \top\big\} \subseteq |D| \times \big(|\Sigma(D,F)|^{\top} \to |\Sigma(\Sigma(D,F),G)|^{\top}\big)$$

is open, as it is a projection of the open set

$$\big\{\langle x, u, g\rangle \;\big|\; g(x,u) = \top\big\} \subseteq |\Sigma(D,F)| \times \big(|\Sigma(D,F)|^{\top} \to |\Sigma(\Sigma(D,F),G)|^{\top}\big) \ .$$

Next we verify that ϕ and θ represent morphisms and that they are inverses of each other. Since we only work with total elements from now on, we do not have to worry about the cases when $\top$ appears as an argument or a result of an application.

(1) ϕ represents a morphism $U \to V$ in the slice over $\mathrm{Q}D$. Let $\langle x, f\rangle, \langle x', f'\rangle \in \|U\|$ and suppose $\langle x, f\rangle \uparrow \langle x', f'\rangle$. This means that $x \uparrow x'$ and $f^{[x\vee x']} \uparrow f'^{[x\vee x']}$, i.e., for every $w \in |F(x \vee x')|$

$$(f(w_{[x]}))^{[\langle x\vee x', w\rangle]} \uparrow (f'(w_{[x']}))^{[\langle x\vee x', w\rangle]} \ .$$

We prove that $\phi(x, f) \approx_V \phi(x', f')$. Clearly, $x \uparrow_D x'$ since $x \uparrow x'$ and $x, x' \in \|D\|$. Let

$$\begin{aligned} g &= \pi_2(\phi(x, f)) = \lambda\langle y, u\rangle \in |\Sigma(D, F)| \,.\, \langle x, t(y, x)u, f(t(y, x)u)\rangle \\ g' &= \pi_2(\phi(x', f')) = \lambda\langle y, u\rangle \in |\Sigma(D, F)| \,.\, \langle x', t(y, x')u, f'(t(y, x')u)\rangle \ . \end{aligned}$$

Let $y, y' \in \|D\|$ such that $y \uparrow y'$ and $y \uparrow x$. Let $u \in \|Fy\|$ and $u' \in \|Fy'\|$ such that $u^{[y\vee y']} \uparrow u'^{[y\vee y']}$. We need to show the following:

(a) $\langle y, u\rangle \uparrow \langle x, t(y, x)u\rangle$
(b) $g(y, u) \in \|\Sigma(\Sigma(D, F), G)\|$
(c) $g'(y', u') \in \|\Sigma(\Sigma(D, F), G)\|$
(d) $(g(y, u))^{[\langle y,u\rangle\vee\langle y',u'\rangle]} \uparrow (g'(y', u'))^{[\langle y,u\rangle\vee\langle y',u'\rangle]}$.

Proof of (a): by assumption $y \uparrow x$, and $u^{[x\vee y]} \uparrow t(y, x)(u)^{[x\vee y]}$ holds because of the common upper bound:

$$\begin{aligned} u^{[x\vee y]} &\le t(y, x \vee y)u \\ (t(y, x)u)^{[x\vee y]} &\le (t(x, x \vee y) \circ t(y, x))u \le t(y, x \vee y)u \ . \end{aligned}$$

Proof of (b): by assumption $x \in \|D\|$, and also $t(y, x)u \in \|Fx\|$ because $x, y \in \|D\|$, $x \uparrow y$ and $u \in \|Fy\|$. Finally, $f(t(y, x)u) \in \|G(x, t(y, x)u)\|$ because $f \in \|\Pi(Fx, \widetilde{G}x)\|$. The proof of (c) is analogous to the proof (b).
Proof of (d): by assumption $x \uparrow x'$, and $(t(y, x)u)^{[x\vee x']} \uparrow (t(y', x')u')^{[x\vee x']}$ holds because

$$\begin{aligned} (t(y, x)u)^{[x\vee x']} &\le t(y, x \vee x')u \le t(y \vee y', x \vee x')(u^{[y\vee y']}) \\ (t(y', x')u')^{[x\vee x']} &\le t(y', x \vee x')u' \le t(y \vee y', x \vee x')(u'^{[y\vee y']}) \end{aligned}$$

and $u^{[y\vee y']} \uparrow u'^{[y\vee y']}$. Let $z = t(y, x)u$ and $z' = t(y', x')u'$, and let $w = z^{[x\vee x']} \vee z'^{[x\vee x']}$. We claim that

$$(fz)^{[\langle x\vee x', w\rangle]} = (fz)^{[\langle x,z\rangle\vee\langle x',z'\rangle]} \uparrow (f'z')^{[\langle x,z\rangle\vee\langle x',z'\rangle]} = (f'z')^{[\langle x\vee x', w\rangle]} \ .$$

From $z \le w_{[x]}$ it follows that $fz \le f(w_{[x]})$, hence

$$(fz)^{[\langle x\vee x', w\rangle]} \le (f(w_{[x]}))^{[\langle x\vee x', w\rangle]} \ ,$$

and similarly,

$$(f'z')^{[\langle x\vee x',w\rangle]} \leq (f'(w_{[x']}))^{[\langle x\vee x',w\rangle]} \ .$$

The claim holds because $f(w_{[x]})^{[\langle x\vee x',w\rangle]} \uparrow f'(w_{[x']})^{[\langle x\vee x',w\rangle]}$.

(2) θ represents a morphism $V \to U$ in the slice over $\mathrm{Q}D$. The proof goes along the same lines as the proof of (1) and is omitted.

(3) $\theta \circ \phi \approx_{U\to U} 1_U$. Let $\langle x, f\rangle \in \|U\|$. We need to show that $\theta(\phi(x,f)) \uparrow \langle x, f\rangle$. The first component is obvious since $\pi_1(\theta(\phi(x,f))) = x$. As for the second component, for any $v \in \|Fx\|$,

$$\begin{aligned}(\pi_2(\theta(\phi(x,f))))v &= s(\langle x, t(x,x)v\rangle, \langle x, v\rangle)(f(t(x,x)v)) \\ &\geq s(\langle x, v\rangle, \langle x, v\rangle)(fv) \\ &\geq fv \ ,\end{aligned}$$

hence $\pi_2(\theta(\phi(x,f))) \uparrow f$.

(4) $\phi\circ\theta \approx_{V\to V} 1_V$. Let $\langle x, g\rangle \in \|V\|$. We need to show that $\phi(\theta(x,g)) \approx_V \langle x, g\rangle$. Again, the first component is obvious since $\pi_1(\phi(\theta(x,g))) = x$. For the second component, given any $\langle y, u\rangle \in \|\Sigma(D, F, \|)$ such that $x \uparrow y$, what has to be shown is

$$\langle x, t(y,x)u, s(g_1(x,t(y,x)u), \langle x, t(y,x)u\rangle)(g_2(x,t(y,x)u))\rangle \uparrow g(y,u) \ .$$

First, we have

$$\langle x, t(y,x)u\rangle \uparrow \langle y, u\rangle \text{ and } \langle y, u\rangle \uparrow g_1(y,u),$$

and since these are elements of a codense totality, we may conclude by transitivity that $\langle x, t(y,x)u\rangle \uparrow g_1(y,u)$. Let $z = g_1(y,u)$ and $w = \langle x, t(y,x)u\rangle$. The relation

$$(g_2(y,u))^{[z\vee w]} \uparrow (s(g_1w, w)(g_2w))^{[z\vee w]}$$

holds because

$$\begin{aligned}(g_2(y,u))^{[z\vee w]} &\leq s(z, z\vee w)(g_2(y,u)) \\ s(g_1w, w)(g_2w)^{[z\vee w]} &\leq s(g_1w, z\vee w)(g_2w) \ ,\end{aligned}$$

and $(y,u) \uparrow w$ together with monotonicity of the function $s(g_1\Box, z\vee w)(g_2\Box)$ imply that

$$s(z, z\vee w)(g_2(y,u)) \uparrow s(g_1w, z\vee w)(g_2w) \ .$$

This concludes the proof of the Main Theorem.

Let $\mathcal{B}$ be the full subcategory of Equ on objects $\mathrm{Q}D$ where D is a natural totality, i.e., $\|D\|$ is a dense, codense, and upward closed subset of $|D|$. It is the case that $\mathcal{B}$ is a cartesian closed subcategory of Equ, see [2]. However, note that the Main Theorem does not imply that $\mathcal{B}$ is a locally cartesian closed subcategory of Equ. We only showed that $\mathcal{B}$ is closed under those dependent sums and products that correspond to parametrizations on domains. In order to resolve the question whether $\mathcal{B}$ is locally cartesian closed it would be useful to have a good characterization of $\mathcal{B}$ in terms of the categorical structure of Equ.

6 Continuous Choice Principle

As an application of the Main Theorem, we translate Berger's Continuous Choice Principle for dependent totalities [4] into a Choice Principle expressed in the *internal* logic of Equ. The internal logic of Equ is a predicative version of intuitionistic first-order logic with dependent types, subset types, and regular quotient types. It is the logic that Equ inherits as a subcategory of the realizability topos $\mathsf{RT}(\mathcal{P}\omega)$, see [6] for details. In this section we use obvious and customary notational simplifications for dependent products and sums.

Let (D, F) be a dependent totality. By [4, Proposition 3.5.2] there is a continuous functional

$$\mathsf{choose} \in |\Pi(x : D, (Fx \to \mathbb{B}_\perp) \to Fx)|$$

such that for all $x \in \|D\|$ and $p \in \|Fx \to \mathbb{B}\|$, if $p^*(\mathsf{true}) \neq \emptyset$, then $(\mathsf{choose}\, x)p \in p^*(\mathsf{true}) \cap \|Fx\|$. Let $X = \mathrm{Q}D$, $Y = \mathrm{Q}(D, F)$ and $2 = \mathrm{Q}(\mathbb{B}_\perp)$. By looking at the proof of [4, Proposition 3.5.2], we see that choose is *not* a *total* functional of type $\|\Pi(x : D, (Fx \to \mathbb{B}_\perp) \to Fx)\|$ because choose applied to the constant function $\lambda x.\, \mathsf{false}$ yields $\perp$, which is not total. This means that choose does not represent a morphism in Equ. Nevertheless we can use it to construct a realizer for the following Choice Principle, stated in the internal logic of Equ:

$$\forall\, p \in (\textstyle\sum_{x:X} Yx) \to 2\,.\, \Big(\big(\forall\, x \in X\,.\, \neg\neg\exists\, y \in Yx\,.\, (p(x,y) = \mathsf{true})\big) \implies \big(\exists\, h \in \textstyle\prod_{x:X} Yx\,.\, \forall\, x \in X\,.\, p(x, hx) = \mathsf{true}\big)\Big) \tag{7}$$

We omit the proof. Suffice it to say that (7) is realized using choose in much the same way as in the proof of [4, Corollary 3.5.3].

If we specialize (7) by setting $X = 1$ and $Y = \mathbb{N}$, we obtain

$$\forall\, p \in \mathbb{N} \to 2\,.\, \big(\neg\neg\exists\, y \in \mathbb{N}\,.\, py = \mathsf{true}\big) \implies \exists\, z \in \mathbb{N}\,.\, pz = \mathsf{true}\big)$$

This is a form of *Markov's Principle*, see for example [25, Vol. 1, Chap. 4, Sect. 5]. Thus, (7) is a generalization of Markov's Principle. This view is in accordance with the construction of the choose functional in [4], which works by *searching* for a witness.

7 Concluding Remarks

We have shown that dependent sums and dependent products of continuous parametrizations on domains with dense, codense, and natural totalities agree with dependent sums and dependent products in Equ. This subsumes our result from [2] and gives further support to Dana Scott's remark that Equ is a theory of *total* functions. Our result can be combined with the result by Normann and Waggbø, who related dependent types in domains with totality and dependent types in limit spaces [20], and with the results by Rosolini, who related dependent

types in Equ to dependent types in various categories of filter spaces [22]. The conclusion is that the dependent-type hierarchies over the natural numbers agree in four settings: domains with totality, equilogical spaces, and thus also in the realizability topos $\mathsf{RT}(\mathcal{P}\omega)$, limit spaces, and filter spaces.

Once the Main Theorem was established, we could use the Continuous Choice Principle of Berger from the setting of domains with totality to show the validity of a Choice Principle in Equ. The Choice Principle in Equ is most concisely stated in the internal logic of Equ, and it would be interesting to prove it directly in Equ. It is likely that such a proof requires better understanding of what codensity corresponds to in Equ. It is not clear how to express codensity in terms of the categorical or the internal logical structure of Equ. We remark that every dense and codense totality D translates into a totally disconnected equilogical space $\mathrm{Q}D$. An equilogical space X is totally disconnected when the curried form of the evaluation map $X \to 2^{2^X}$ is monic, or equivalently, when the topological quotient $\|X\|/{\approx_X}$ is a totally disconnected space. There are totally disconnected equilogical spaces that do not arise as dense and codense totalities. The subcategory of totally disconnected equilogical spaces is a locally cartesian closed subcategory of Equ. Perhaps the notion of total disconnectedness, or some refinement of it, can be useful for this purpose.

The Main Theorem can be used to infer another consequence about equilogical spaces. Berger [4,5] showed that extensional equality on the dependent-type hierarchy over the natural numbers coincides with the partial equivalence relation induced by the consistency relation on the underlying domains. This is important because the logical complexity of extensional equality is as complicated as the type at which it is defined, whereas consistency can be expressed as a Π^0_1 statement and has bounded logical complexity. The Main Theorem implies an analogous result for equality in Equ.

References

1. R.M. Amadio and P.-L. Curien. *Domains and Lambda-Calculi*, volume 46 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1998.
2. A. Bauer, L. Birkedal, and D.S. Scott. Equilogical spaces. *Preprint submitted to Elsevier*, 1998.
3. U. Berger. Total sets and objects in domain theory. *Annals of Pure and Applied Logic*, 60:91–117, 1993.
4. U. Berger. *Continuous Functionals of Dependent and Transitive Types*. Habilitationsschrift, Ludwig-Maximilians-Universität München, 1997.
5. U. Berger. Continuous functionals of dependent and transitive types. Short version. Available at `http://www.mathematik.uni-muenchen.de/~berger/habil/lc.dvi.Z`., 1997.
6. L. Birkedal. *Developing Theories of Types and Computability*. PhD thesis, School of Computer Science, Carnegie Mellon University, December 1999. Available as CMU Technical Report: CMU-CS-99-173.
7. Y.L. Ershov. Model $\mathbb{C}$ for partial continuous functionals. In *Logic Colloquium 1976*, pages 455–467. North-Holland, 1977.

8. R.O. Gandy and J.M.E. Hyland. Computable and recursively countable functions of higher type. In R.O. Gandy and J.M.E. Hyland, editors, *Logic Colloquium 1976*, pages 407–438, Amsterdam, Holland, 1977. North-Holland.
9. R. Heckmann. On the relationship between filter spaces and equilogical spaces. Available at http://www.cs.uni-sb.de/RW/users/heckmann/domains.html.
10. J.M.E. Hyland. Filter spaces and continuous functionals. *Annals of Mathematical Logic*, 16, 1979.
11. S.C. Kleene. Countable functionals. In *Constructivity in Mathematics*, pages 81–100, 1959.
12. L. Kristiansen and D. Normann. Total objects in inductively defined types. *Arch. Math. Logic*, 36:405–436, 1997.
13. G. Longo and E. Moggi. The heredetarily partial effective functionals and recursion theory in higher types. *Journal of Symbolic Logic*, 40:1319–1332, 1984.
14. M. Menni and A. Simpson. Topological and limit-space subcategories of countably-based equilogical spaces. Submitted to Math. Struct. in Comp. Science. Available at http://www.dcs.ed.ac.uk/home/als/Research/, November 1999.
15. D. Normann. *Recursion on the Countable Functionals*. Number 811 in Lecture Notes in Mathematics. Springer Verlag, 1980.
16. D. Normann. A hierarchy of domains with totality, but without density. In S.B. Cooper, T.A. Slaman, and S.S. Wainer, editors, *Computability, Enumerability, Unsolvability*, pages 233–257. Cambridge University Press, 1996.
17. D. Normann. Closing the gap between the continuous functionals and recursion in ^3E. *Arch. Math. Logic*, 36:269–287, 1997.
18. D. Normann. Categories of domains with totality. Available at http://www.math.uio.no/~dnormann/, June 1998.
19. D. Normann. The continuous functionals. In E.R. Griffor, editor, *Handbook of Computability Theory*. Elsevier, Amsterdam, Holland, 1998.
20. D. Normann and G. Waagbø. Limit spaces and transfinite types. Available at http://www.math.uio.no/~dnormann/, 1998.
21. G. Plotkin. Full abstraction, totality, and PCF. *Mathematical Structures in Computer Science*, 1998.
22. G. Rosolini. Equilogical spaces and filter spaces. Available at ftp://ftp.disi.unige.it/pub/person/RosoliniG/papers/equsfs.ps.gz, 1999.
23. H. Schwichtenberg. Density and choice for total continuous functionals. In P. Odifreddi, editor, *Kreiseliana. About and Around George Kreisel*, pages 335–365. A K Peters, 1996.
24. V. Stoltenberg-Hansen, I. Lindström, and E.R. Griffor. *Mathematical Theory of Domains*. Number 22 in Cambridge Tracts in Computer Science. Cambridge University Press, 1994.
25. A.S. Troelstra and D. van Dalen. *Constructivism in Mathematics*. North-Holland, 1988. 2 volumes.
26. G. Waagbø. *Domains-with-totality Semantics for Intuitionistic Type Theory*. Dr. scient. thesis, The University of Oslo, 1997.

Definability over Linear Constraints

Michael Benedikt[1] and H. Jerome Keisler[2]

[1] Bell Laboratories, 263 Shuman Blvd., Naperville, IL 60566, USA,
benedikt@research.bell-labs.com
[2] University of Wisconsin, Madison Wisconsin 53706, USA,
keisler@math.wisc.edu

Abstract. We settle a number of questions concerning definability in first order logics with an extra predicate symbol ranging over semi-linear sets. These questions are motivated by the constraint database model for representing spatial data. We give new results both on the positive and negative side: we show that in first-order logic one cannot query a semi-linear set as to whether or not it contains a line, or whether or not it contains the line segment between two given points. However, we show that some of these queries become definable if one makes small restrictions on the semi-linear sets considered.

1 Introduction

Much recent work in the foundations of spatial databases concerns the modeling of spatial information by *constraint sets*: Boolean combinations of linear or polynomial inequalities. Constraint sets can be effectively queried using variants of first-order logic; this is the basic idea behind *constraint query languages* ([9], [7]). The most well-studied languages in this family are the *first-order linear constraint language* FO_{LIN} and the *first-order polynomial constraint language* FO_{POLY}. By a *semi-linear set* we mean a subset of a Euclidean space R^n which is definable by a linear constraint, that is, a quantifier-free first order formula in the real ordered group $\langle R, +, -, < \rangle$. (Here and throughout this paper, formulas may have parameters from R). By quantifier elimination, each first order formula is equivalent to a linear constraint in the real ordered group. FO_{LIN} is the first-order language with the vocabulary of the real ordered group plus an extra predicate symbol S which ranges over semi-linear sets. Every FO_{LIN} sentence defines a collection of semi-linear sets. In FO_{POLY}, the product symbol $\times$ is added to the vocabulary, and the extra predicate symbol S ranges over the *semi-algebraic sets*— the subsets of R^n which are definable by polynomial constraints, i.e. quantifier-free (or first order) formulas in the ordered field of reals.

A basic question, then, concerns the expressive power of these languages. Which families of definable sets (semi-linear sets for FO_{LIN}, semi-algebraic sets for FO_{POLY}) can be defined by a sentence in the language of the real ordered

P. Clote and H. Schwichtenberg (Eds.): CSL 2000, LNCS 1862, pp. 217–232, 2000.

group (respectively field) with an extra predicate symbol ranging over the definable sets? More generally, given a family of sets F in Euclidean space, one can ask: Which subfamilies of F can be defined by a sentence in FO_{LIN} or FO_{POLY} with an extra predicate symbol ranging over F. Recent work has clarified many questions about the expressive power of FO_{LIN} and FO_{POLY} with an extra predicate symbol ranging over the *finite* subsets of Euclidean space ([3], [9]). There are also a number of recent results about the expressiveness of FO_{POLY} with an extra predicate symbol ranging over the semi-algebraic sets ([5], [8]). However, the expressiveness of FO_{LIN} with an extra predicate symbol ranging over the semi-linear sets is much less understood. Let's consider the following examples in the Euclidean plane:

$$Colinear = \{A \subset R^2 : \text{ all points in } A \text{ are colinear } \}$$

$$Is.Line = \{A \subset R^2 : A \text{ is the graph of some line } \}$$

$$Cont.Line = \{A \subset R^2 : A \text{ contains the graph of some line } \}$$

$$Lin.Reach = \left\{ (A, \boldsymbol{a}, \boldsymbol{b}) : \begin{array}{l} A \subset R^2, \boldsymbol{a}, \boldsymbol{b} \in R^2, \\ A \text{ contains the line segment from } \boldsymbol{a} \text{ to } \boldsymbol{b} \end{array} \right\}$$

In the last example, FO_{LIN} has extra constant symbols ranging over R, in addition to the extra predicate symbol S. Each of the four examples is easily seen to be definable by a sentence in FO_{POLY}, since there one can quantify over lines. Are they also definable in FO_{LIN}? It is fairly straightforward to show that *Colinear* is *not* definable by a sentence in FO_{LIN} (see [1] and remark 1 below). However *Is.Line* is definable in FO_{LIN} [2]. A semi-linear set A belongs to the collection *Is.Line* iff it is either a vertical line or is the graph of a function and has the property that if $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \in A$ then $\boldsymbol{x} + (\boldsymbol{y} - \boldsymbol{z}) \in A$.

It was asked in [1] whether *Cont.Line* is definable in FO_{LIN}. This and related questions were also considered in [2]. The general question is: under what circumstances can we ask questions about the existence of lines or line-segments in FO_{LIN}? If it appears that a query cannot be expressed in FO_{LIN}, how can we prove this? There is a need for techniques to show that a family of semi-linear sets is or is not definable in FO_{LIN}. In this paper we introduce such techniques, particularly methods from nonstandard analysis, and use them to resolve numerous questions about the definability of sets such as *Cont.Line* and *Lin.Reach*. We also show some positive results, giving that over certain classes of sets queries with quantification over lines are expressible. For example, we show that *Cont.Line* is undefinable over arbitrary semi-linear sets, but we also show that *Cont.Line* is definable if the extra predicate symbol ranges over an interesting subclass, the thin semi-linear sets. We also investigate several natural languages between FO_{LIN} and FO_{POLY}. We give game characterizations of definability in these languages, and extend several of the undefinability results.

Organization: Section 2 introduces the notation and basic definitions. Section 3 gives the basic negative results showing that a number of queries are undefinable in FO_{LIN}. Section 4 gives positive results showing that certain queries are

definable when the extra predicate S ranges over restricted classes of semi-linear sets. Section 5 is about the definability of the property that any two points are n-linked (connected by a polygonal path with at most n segments). Conclusions are given in Section 6.

2 Notation

2.1 The Language FO_{LIN}

We start with a signature $\mathcal{S}$ consisting of predicate and constant symbols. For simplicity we will confine our attention to the case where $\mathcal{S} = \langle S, \boldsymbol{c} \rangle$ has only one binary predicate symbol S, and a sequence $\boldsymbol{c}$ of constant symbols of length l. The sequence $\boldsymbol{c}$ of constant symbols may be empty, that is, $l = 0$.

We let $FO_{LIN}(\mathcal{S})$ be the first-order language over the vocabulary $\mathcal{S} \cup \{+, -, <\}$. When $\mathcal{S}$ is clear from context we refer simply to FO_{LIN}. The first order structures for this vocabulary have the form $\langle R, +, -, <, A, \boldsymbol{a} \rangle$ where $A \subset R^2$ interprets S and $\boldsymbol{a} \in R^l$ interprets $\boldsymbol{c}$. Since all the structures under consideration have the same $\langle R, +, -, < \rangle$ part, we concentrate on the other part and define an $\mathcal{S}$-structure to be an object $\mathcal{A} = \langle A, \boldsymbol{a} \rangle$ where $A \subset R^2$ interprets S and $\boldsymbol{a} \in R^l$ interprets $\boldsymbol{c}$.

An $\mathcal{S}$-structure $\mathcal{A} = \langle A, \boldsymbol{a} \rangle$ *satisfies* a sentence $\phi \in FO_{LIN}(\mathcal{S})$, in symbols $\mathcal{A} \models \phi$, exactly when the corresponding first order structure $\langle R, +, -, <, \mathcal{A} \rangle = \langle R, +, -, <, A, \boldsymbol{a} \rangle$ satisfies ϕ. If the relation A is semi-linear, we say that $\mathcal{A}$ is a *semi-linear structure*, or *semi-linear instance*. Semi-algebraic and semi-analytic structures are defined similarly. Any collection of $\mathcal{S}$-structures is called a (Euclidean) *query*. By a *semi-linear query*, we will mean a collection of semi-linear structures. We say that a query X is FO_{LIN}*-definable* if there is an FO_{LIN} sentence ϕ such that:

$$\text{for every semi-linear structure } \mathcal{A}, \quad \mathcal{A} \models \phi \text{ if and only if } \mathcal{A} \in X.$$

We will sometimes consider the following generalization where the family of all semi-linear structures is replaced by another family of structures. Given a "base" query F (not necessarily semi-linear), we say that a query X is FO_{LIN}*-definable over* F if there is a sentence ϕ of FO_{LIN} such that:

$$\text{for every } \mathcal{A} \in F, \quad \mathcal{A} \models \phi \text{ if and only if } \mathcal{A} \in X.$$

Thus when a base F is not mentioned, it is understood to be the family of all semi-linear structures. Some examples of base queries F that will arise in this paper are the families of semi-linear structures $\mathcal{A} = \langle A, \boldsymbol{a} \rangle$ such that the relation A is:

- finite;
- *thin*, that is, has an empty interior;
- *1-bounded*, that is, is contained in a unit square $[0,1]^2$;
- has at most n *singular points* (a singular point is a vertex of the boundary of A).

We note that if a query is FO_{LIN} definable over F then it is FO_{LIN} definable over any subcollection $E \subseteq F$. Thus definability results are stronger when F is larger, while undefinability results are stronger when F is smaller. We will study the definability of some natural queries in FO_{LIN} and its extensions. As a starting point, we recall a known fact, which is a consequence of the classical result that the product function cannot be defined in the first order theory of $\langle R, +, -, < \rangle$.

Remark 1. The query *Colinear* is not FO_{LIN} definable over the collection of subsets of the plane of cardinality three.

2.2 Nonstandard Analysis and Undefinability

We will use notions from nonstandard analysis as a tool in many of our proofs. However, all of our results are statements about the standard reals.

The main place where nonstandard notions will be used is in characterizations of definability in query languages (as in, e.g., [4]). We assume familiarity with basic notions of nonstandard analysis (see [6]), but give a briefer-than-brief review here.

N denotes the set of positive integers. For any set U, the *superstructure* $V(U)$ with base set U is defined as $V(U) = \bigcup_{n \in N} V_n(U)$ where $V_1(U) = U$, and $V_{n+1}(U) = V_n(U) \cup \{X : X \subset V_n(U)\}$. Note in particular that $U \in V(U)$. We will work with the superstructure $\langle V(U), \in \rangle$ considered as a structure for the first-order language with the binary relation $\in$. A *bounded quantifier formula* in this language is a formula built up from atomic formulas by the logical connectives and the bounded quantifiers: $\forall X \in Y$, $\exists X \in Y$, where X and Y are variables. Almost all of "classical" mathematics can be done within the superstructure $V(R)$ based on the set R of reals.

A *nonstandard universe* (based on R) consists of a pair of superstructures $V(R)$ and $V(^*R)$ and a mapping $* : V(R) \to V(^*R)$ such that:

1. $^*R$ is a proper extension of R
2. For each $r \in R$, $^*r = r$
3. **(Transfer Principle)** For any bounded quantifier formula $\phi(v_1, \ldots, v_n)$ and any list $a_1, \ldots, a_n$ of elements from $V(R)$, $\phi(a_1, \ldots, a_n)$ is true in $V(R)$ if and only if $\phi(^*a_1, \ldots, ^*a_n)$ is true in $V(^*R)$.

We will fix a nonstandard universe once and for all.

Note that ${}^*R$ is the image of the element $R \in V(R)$, and ${}^*R \in V({}^*R)$. An element $B \in V({}^*R)$ is *standard* if it is in the image of the $*$-map, that is, $B = {}^*A$ for some $A \in V(R)$, and *internal* if it is an element of a standard set, that is, $B \in {}^*A$ for some $A \in V(R)$.

Some examples of standard sets are ${}^*R$, the usual order relation and arithmetic operations on ${}^*R$, and the sets ${}^*Z$, ${}^*Q$, and ${}^*N$. To improve readability, we ordinarily drop the $*$ from the order relation and arithmetic operations of ${}^*R$.

All standard sets are internal, all elements of internal sets internal, and any finite subset of an internal set is internal. Other examples of internal sets are the closed intervals ${}^*[a, b]$ where $a, b \in {}^*R$, and more generally the sets and relations which are first-order definable in the structure $\langle\, {}^*R, {}^*N, <, +, -, \times \,\rangle$.

An element $r \in {}^*R$ is *finite* if $|r| < n$ for some $n \in N$, and *infinitesimal* if $|r| < 1/n$ for all $n \in N$. For $r, s \in {}^*R$, we write $r \approx s$ if $|r - s|$ is infinitesimal. For each finite $r \in {}^*R$, there is a unique standard real number ${}^o r \in R$, called the *standard part* of r, such that ${}^o r \approx r$.

Three important consequences of the definition are:

- N is a proper initial segment of ${}^*N$ in the natural ordering.
- Every nonempty *internal* subset of ${}^*R$ which has an upper bound has a least upper bound.
- Every infinite internal set is uncountable (and has cardinality at least the continuum).

It follows that infinite and positive infinitesimal elements of ${}^*R$ exist. In fact, there are uncountably many infinite $K \in {}^*N$, and uncountably many infinitesimals in ${}^*Q$.

Some examples of sets in $V({}^*R)$ which are not internal are: any nonempty subset of ${}^*R$ which has an upper bound but no least upper bound (such as R, the set of finite elements, or the set of infinitesimals), any countably infinite set, the set of all finite subsets of ${}^*R$, and the standard part function ^{o}.

By the Transfer Principle, the mapping $*$ is an elementary embedding of the ordered ring $\langle\, Z, +, -, \times, < \,\rangle$ into $\langle\, {}^*Z, +, -, \times, < \,\rangle$, and similarly for R and ${}^*R$. Many of the facts we need from nonstandard analysis can be derived from these elementary embedding results.

When a set $A \in V(R)$ has a name, say the set of *widgets*, the elements of ${}^*A$ are called **widgets*, or *hyperwidgets*. For example, ${}^*R$ is the set of *hyperreal numbers*, and the image of the collection of semi-linear sets is the collection of *hypersemi-linear sets*. Thus every hypersemi-linear set is internal. Hypersemi-linear sets will appear in many of our proofs. When discussing properties of a hypersemi-linear set, we will often drop the "hyper" prefix; for example, we will usually write "line" rather than "hyperline", and "connected" rather than "hyperconnected".

By the Transfer Principle, any set which is definable by a first order formula in $\langle {}^*R, +, -, < \rangle$ is hypersemi-linear. In fact, any set which is definable by a first order hyperformula with parameters in $\langle {}^*R, +, -, < \rangle$ (or, equivalently, by a hyperfinite Boolean combination of linear constraints) is still hypersemi-linear.

The following proposition will be useful in proving undefinability results. In this proposition, $\equiv$ stands for the elementary equivalence relation between first order structures.

Proposition 1. *Suppose X and F are Euclidean queries. Then the following are equivalent:*

1. *X is not FO_{LIN}-definable over F.*
2. *There are hyperstructures $\mathcal{A}, \mathcal{B} \in {}^*F$ such that $\mathcal{A} \in {}^*X$ and $\mathcal{B} \notin {}^*X$, but*

$$\langle {}^*R, +, -, <, \mathcal{A} \rangle \equiv \langle {}^*R, +, -, <, \mathcal{B} \rangle.$$

Proof: We will only use the direction from 2 to 1 in this paper, so we prove that direction here and leave the converse as an exercise. Assume that 1 fails but 2 holds. Let ϕ be an FO_{LIN} sentence which defines X over F. By the Transfer Principle, since $\mathcal{A} \in {}^*X$, we have $\mathcal{A} \models \phi$ and $\langle {}^*R, +, -, <, \mathcal{A} \rangle \models \phi$. Similarly, since $\mathcal{B} \notin {}^*X$, we have $\mathcal{B} \models \neg\phi$ and $\langle {}^*R, +, -, <, \mathcal{B} \rangle \models \neg\phi$. This contradicts 2. □

3 Undefinability in First-Order Logic

Theorem 1. *The query Lin.Reach is not definable in FO_{LIN}.*

Proof: To do this we construct a hypersemi-linear set A as follows. Let δ be a positive infinitesimal and m be an element of ${}^*(0,1)$ which satisfy requirements to be given below. Let B be the set of parallel lines $y = K\delta + mx$, with $K \in {}^*Z$, and let A be the intersection of B with the hyperreal unit square ${}^*[0,1]^2$. Thus δ will be the vertical distance between line segments, and m will be the slope of any of the lines. See Figure 1.

Let $D = \{K : K/n \in {}^*Z \text{ for all } n \in N\}$. D is the largest divisible subgroup of $\langle {}^*Z, +, - \rangle$. D is nontrivial, since $K! \in D$ for any infinite $K \in {}^*N$. Choose hyperrational numbers $m, \delta \in {}^*Q \cap {}^*[0,1]$ so that:

- $m = J/L \in {}^*Q$ where $J, L \in {}^*N$ and $J \leq L$
- The standard part $^\circ m$ (which belongs to $[0,1]$) is irrational
- $H = 1/\delta$ is in D and is such that $H/L \in D$ (i.e. H is L times something in D)

Our aim is to find a function f satisfying the following requirements. Let $\Delta(x) = f(x) - x$. We say that a function $f : {}^*R \to {}^*R$ is *good* if for all $x, y \in {}^*R$:

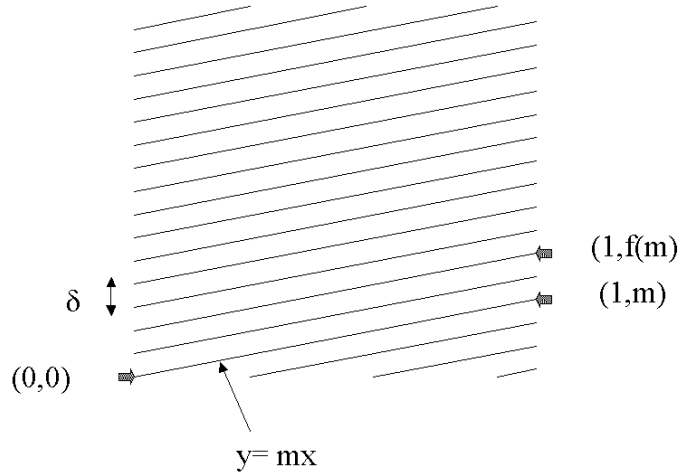


Fig. 1. The Basic Construction

- f is a bijection
- f is order preserving
- f is linear, i.e. $f(x+y) = f(x) + f(y)$
- $f(\epsilon) = \epsilon$ for all infinitesimal ϵ
- $f(z) = z$ for all $z \in {}^*Z$
- $f(x) \approx x$ (hence $\Delta(x) \approx 0$)
- $\Delta(x)/\delta \in D$ and $\Delta(x)m/\delta \in D$
- $f(m) \neq m$

For any function f on the hyperreal line, we let f^2 be the map on the plane defined by $f^2((x,y)) = (f(x), f(y))$.

Claim 1 *For any good function f, f^2 maps any point lying on a line of the form $y = K\delta + mx$, with $K \in {}^*Z$, to a point on another line of this form.*

Proof: Given $y = K\delta + mx$, we show that $(f(x), f(y))$ is of the required form: i.e. $(f(y) - mf(x))/\delta \in {}^*Z$. We have

$f(y) - mf(x) = f(K\delta) + f(mx) - mf(x) = K\delta + \Delta(K\delta) + mx + \Delta(mx) - m(x + \Delta(x))$.

This is $K\delta + \Delta(K\delta) + \Delta(mx) - m\Delta(x)$. But each of these four terms is a multiple of δ, because for every z, $\Delta(z)/\delta \in D \subset {}^*Z$, and similarly, $m\Delta(x)/\delta \in {}^*Z$. □

Lemma 1. *There is a good function f.*

The proof of this lemma will take some time. Let R_0 be the smallest divisible subgroup of $\langle {}^*R, +, - \rangle$ that contains both ${}^*Z$ and the set of all infinitesimals. Note that R_0 is just the set of all $x \in {}^*R$ such that the standard part of the fractional part of x is rational.

Claim 2 *R_0 is equal to the set of all $x \in {}^*R$ such that $x = p + n + \epsilon$ for some $p \in Q \cap [0,1)$, $n \in {}^*Z$, and $\epsilon \approx 0$. Moreover, for each $x \in R_0$, the decomposition $x = p + n + \epsilon$ is unique.*

Proof: The first statement is clear. To prove uniqueness, suppose $p + n + \epsilon = p' + n' + \epsilon'$. Then $n - n' = (p' - p) + (\epsilon' - \epsilon)$. The left side of this equation belongs to ${}^*Z$, and the right side is finite and has standard part in $(-1, 1)$. Therefore both sides of the equation are equal to 0. Therefore $n = n'$ and $p' \approx p$. It follows that $p' = p$ and hence $\epsilon' = \epsilon$. □

Now let $\mathbf{c}$ be the cardinality of the continuum. As usual, we identify $\mathbf{c}$ with the set of all ordinals of cardinality less than $\mathbf{c}$. Let $\{r_\alpha : \alpha < \mathbf{c}\}$ be an enumeration of the set of all reals in $[0,1)$. Starting with R_0 defined above, we build an increasing chain of sets R_α, $\alpha < \mathbf{c}$ by the following transfinite recursion. For limit ordinals $\gamma < \mathbf{c}$, put $R_\gamma = \bigcup_{\beta<\gamma} R_\beta$. For successor ordinals $\alpha + 1$, let $R_{\alpha+1}$ be the smallest divisible subgroup of $\langle {}^*R, +, - \rangle$ that contains both R_α and r_α. We then have ${}^*R = \bigcup_{\alpha<\mathbf{c}} R_\alpha$. Note that in the case that $r_\alpha \in R_\alpha$, $R_{\alpha+1}$ is just R_α.

Claim 3 *$R_{\alpha+1}$ is equal to the set of all $x \in {}^*R$ such that $x = pr_\alpha + n + y$ for some $p \in Q$, $n \in {}^*Z$, and $y \in {}^*[0,1) \cap R_\alpha$. Moreover, if $r_\alpha \notin R_\alpha$, then for each $x \in R_{\alpha+1}$ the decomposition $x = pr_\alpha + n + y$ is unique.*

Proof: To prove uniqueness, suppose $pr_\alpha + n + y = p'r_\alpha + n' + y'$. If $p \neq p'$, then $r_\alpha = ((n - n') + (y - y'))/(p' - p) \in R_\alpha$. On the other hand, if $p = p'$, then $n + y = n' + y'$, so $n - n' = y' - y$. The left side of this equation belongs to ${}^*Z$ and the right side belongs to ${}^*(-1, 1)$. Therefore both sides are equal to 0, so $n = n'$ and $y = y'$. □

Claim 4 *For each $\alpha < \mathbf{c}$, R_α contains fewer than $\mathbf{c}$ real numbers.*

Proof: We will prove by transfinite induction that for each $\alpha < \mathbf{c}$, $|R_\alpha \cap R| \leq \aleph_0 + |\alpha| < \mathbf{c}$. By Claim 2, $R_0 \cap R = Q$, so $|R_0 \cap R| = \aleph_0$. If α is a limit ordinal, then by inductive hypothesis, $|R_\alpha \cap R| \leq \sum_{\beta<\alpha}(\aleph_0 + |\beta|) = \aleph_0 + |\alpha|$.

Now assume the result for α. By Claim 3 we have $|R_{\alpha+1} \cap R| \leq |Q| \times |R_\alpha \cap R| \leq \aleph_0 + |\alpha|$. Thus the result holds for $\alpha + 1$, and the induction is complete. □

Claim 5 *There exist infinitely many positive infinitesimal $\epsilon \in {}^*Q$ such that $\epsilon/\delta \in D$ and $\epsilon m/\delta \in D$.*

Proof: By hypothesis, $1/\delta = H$ where $H \in D$ and $H/L \in D$, so $mH = J(H/L) \in D$. Thus for all $n \in N$, $H/n \in {}^*Z$ and $mH/n \in {}^*Z$. Using the Transfer Principle,

there is a least $n \in {}^*N \setminus N$ such that $H/n \notin {}^*Z$ or $mH/n \notin {}^*Z$. Therefore for all sufficiently small infinite $K \in {}^*N$ we have $H/K \in D$ and $mH/K \in D$. Thus $\epsilon = 1/K$ has the required property. □

With another transfinite recursion, we build an increasing chain of functions $f \restriction R_\alpha : R_\alpha \to {}^*R$, $\alpha < \mathbf{c}$. Let $f \restriction R_0$ be the identity function on R_0, take unions at limit ordinals, and define $f \restriction R_{\alpha+1}$ as follows when $r_\alpha \notin R_\alpha$: By Claim 5 we may choose a positive infinitesimal $\epsilon_\alpha \in {}^*Q$ such that $\epsilon_\alpha H \in D$ and $\epsilon_\alpha mH \in D$. For $x \in R_{\alpha+1}$, put $x = pr_\alpha + n + y$ as in Claim 3, and define $f(x) = p(r_\alpha + \epsilon_\alpha) + f(n+y)$. When $x \in R_\alpha$, we have $p = 0$, so the new value of $f(x)$ agrees with the old. Taking the union, we have a function $f : {}^*R \to {}^*R$.

We now add one more requirement in the construction which will insure that $f(m) \neq m$. Consider the first β such that $m \in R_\beta$. Since ${}^o m$ is irrational, $m \notin R_0$, so $\beta > 0$. Then β must be a successor ordinal, $\beta = \alpha + 1$. Since $m \notin R_\alpha$ we have $R_{\alpha+1} \neq R_\alpha$ and thus $r_\alpha \notin R_\alpha$. We then have a unique decomposition $m = pr_\alpha + n + y$ with $p \neq 0$. Any two different choices of the infinitesimal ϵ_α will result in different values for $f(m)$, so we can choose ϵ_α in such a way that $f(m) \neq m$.

Claim 6 *The function f is good.*

Proof: We verify the requirement that $\Delta(x)/\delta \in D$ and $\Delta(x)m/\delta \in D$ for all $x \in {}^*R$. Let $\beta = \alpha + 1$ be the first ordinal such that $x \in R_\beta$, and put $x = pr_\alpha + n + y$. We argue by induction on α. We have $\Delta(x) = f(x) - x = p\epsilon_\alpha + \Delta(n+y)$, where $n + y \in R_\alpha$ and $p \in Q$. By definition, $\epsilon_\alpha/\delta \in D$, and by inductive hypothesis, $\Delta(n+y)/\delta \in D$. Since D is a divisible group, it follows that $\Delta(x)/\delta \in D$. The proof that $\Delta(x)m/\delta \in D$ is similar. The other requirements on f are easily proved by induction on α. □

This completes the proof of Lemma 1.

We now use the good function f to complete the proof of Theorem 1. Consider the hypersemi-linear structures $\mathcal{A} = \langle A, 0, 0, 1, m \rangle$ and $\mathcal{B} = \langle A, 0, 0, 1, f(m) \rangle$. The two points $(0,0), (1,m)$ are on the same line segment in A, so *Lin.Reach* holds in $\mathcal{A}$. But $f(m) \neq m$, so *Lin.Reach* fails in $\mathcal{B}$. Since f is good, it is an automorphism of the hyperreal ordered group that maps the relation and constants in $\mathcal{A}$ to those in $\mathcal{B}$. Hence
$\langle {}^*R, +, -, <, \mathcal{A} \rangle \equiv \langle {}^*R, +, -, <, \mathcal{B} \rangle$. By Proposition 1, *Lin.Reach* is not FO_{LIN} definable. □

The above proof shows more. Let *Conn* be the query

$$Conn = \{(A, \boldsymbol{a}, \boldsymbol{b}) : A \subset R^2, \boldsymbol{a}, \boldsymbol{b} \in R^2, \boldsymbol{a} \text{ is connected to } \boldsymbol{b} \text{ in } A\}.$$

Corollary 1. *Lin.Reach is not FO_{LIN}-definable over the collection F of 1-bounded thin semi-linear sets. In fact, there is no sentence ψ of $FO_{LIN}(\mathcal{S})$ such that $(Lin.Reach \to \psi) \wedge (\psi \to Conn)$ holds in all $\mathcal{A} \in F$.*

Proof: The graph A is 1-bounded and thin. The points $(0,0),(1,m)$ are on the same line segment in the structure $\mathcal{A}$, but the corresponding points $(0,0)$, $(1,f(m))$ are not even connected in the structure $\mathcal{B}$. Thus ψ would have to hold in $\mathcal{A}$ but fail in $\mathcal{B}$, so ψ cannot be a sentence of FO_{LIN}. □

The proof of Theorem 1 can be modified to show that *Lin.Reach* is FO_{LIN} undefinable over other families. For example, the preceding proof uses hypersemi-linear sets with infinitely many parallel line segments, which corresponds to a family of semi-linear sets with unboundedly many parallel line segments, and thus unboundedly many singular points. The next result has a finite bound on the number of singular points but a nonempty interior.

Theorem 2. *Lin.Reach is not FO_{LIN}-definable over the collection of 1-bounded semi-linear sets with ten singular points.*

Proof: We modify the set A to form two new sets T and U (see Figure 2). To form T, we first replace the unit square by the "unit right triangle" with vertices $(0,0),(1,0),(1,1)$. Then we add all of the interior of the unit right triangle except for two similar infinitesimal triangular windows around the points $(0,0)$ and $(1,m)$ with height $< \delta$. These windows are so small that they miss all of the original family of parallel lines of slope m except for the line segment through $(0,0),(1,m)$. The set T formed in this way is a 1-bounded hypersemi-linear set with ten singular points. The set U is similar except that the second triangular window is around the point $(1,f(m))$. We use the same function f as before. f is an isomorphism between the structures $\langle T,0,0,1,m\rangle$ and $\langle U,0,0,1,f(m)\rangle$. The points $(0,0),(1,m)$ are on a line segment contained in T but the points $(0,0),(1,f(m))$ are not on a line segment contained in U. □

We will now show that the query *Cont.Line* is not first order definable. Here the signature has a binary predicate symbol but no constants.

Theorem 3. *The query Cont.Line is not definable in FO_{LIN}. In fact, it's not even FO_{LIN}-definable over the collection of semi-linear sets with ten singular points.*

Proof: We modify the construction in the previous theorem. Let T,U, and f be as in the proof of Theorem 2. We will add two infinite cones to T as follows. Let C_1 be the cone heading towards $-\infty$ with apex $(0,0)$ and bounded by two rays having standard rational slopes r and s where $0 < r < {}^\circ m < s < 1$. See Figure 3.

C_1 is a standard cone defined by rational numbers, and hence will be fixed by f. Let C_2 be the cone with apex $(1,m)$ and bounded by two lines having rational slopes r' and s' with $0 < r' < {}^\circ m < s' < 1$. The base of C_2 will be moved by f, and the rationality of the slopes will guarantee that the boundary lines map to boundary lines. Thus C_2 maps under f to the cone C_2' with apex $(1,f(m))$ and boundary rays having slopes s' and t'. Let $T_1 = T \cup C_1 \cup C_2$ and $U_1 = U \cup C_1 \cup C_2'$. These sets again have ten singular points. T_1 contains a

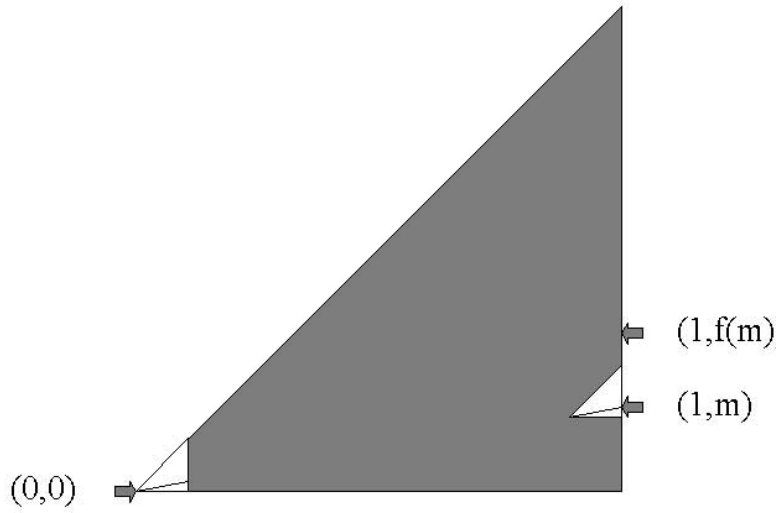


Fig. 2. Modified Construction

line and T_2 does not. It is clear that f maps T_1 to T_2, which shows that the corresponding structures are equivalent in FO_{LIN}. □

4 Definability over Thin Sets

There is an intuition that queries such as *Lin.Reach* and *Cont.Line* are "almost" first-order definable. We will give two positive results showing that *Cont.Line* and *Lin.Reach* are FO_{LIN} definable over restricted classes of semi-linear structures.

We note that in Theorem 3 we made use of examples that had nonempty, and in fact unbounded, interiors. We show here that this is essential. We show that *Cont.Line* is definable over thin semi-linear instances: i.e. there is an FO_{LIN} sentence ϕ such that for each thin semi-linear structure $\mathcal{A}$, $\mathcal{A} \models \phi \leftrightarrow \mathcal{A} \in$ *Cont.Line*.

Theorem 4. *Cont.Line is FO_{LIN} definable over the family of thin semi-linear sets.*

Proof: A point is called *regular* in S if it is not singular in S. If $U(\boldsymbol{y}, \boldsymbol{z})$ denotes the open rectangle with corners $\boldsymbol{y}, \boldsymbol{z}$, then $\boldsymbol{x} \in U(\boldsymbol{y}, \boldsymbol{z})$ can be expressed by the formula $y_1 < x_1 < z_1 \wedge y_2 < x_2 < z_2$. Thus if S is thin one can say that $\boldsymbol{x}$ is regular in S with the formula $Reg(\boldsymbol{x})$:

$\exists \boldsymbol{y} \exists \boldsymbol{z} \exists \boldsymbol{u} [\boldsymbol{u} \neq \boldsymbol{0} \wedge (S \cap U(\boldsymbol{y}, \boldsymbol{z}))$ contains $\boldsymbol{x}+\boldsymbol{u}, \boldsymbol{x}-\boldsymbol{u}$ and is closed under midpoints].

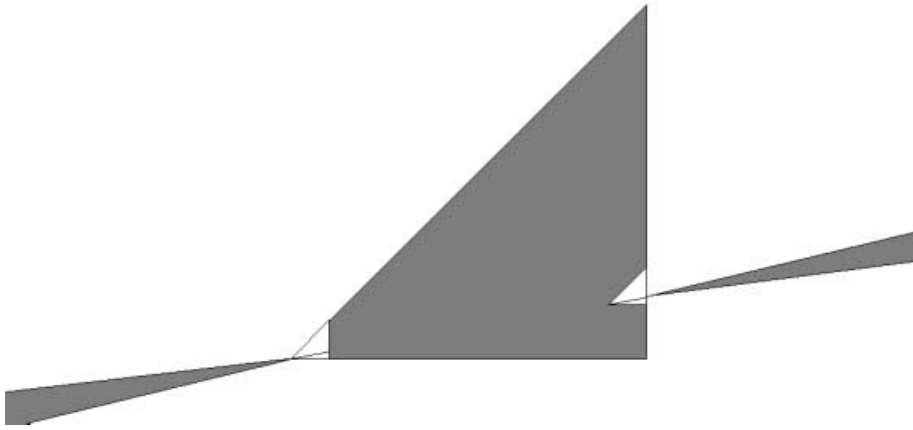


Fig. 3. Contains a line

Let $Cong(\boldsymbol{x}, \boldsymbol{y})$ (for congruence) say that $\boldsymbol{x}$ and $\boldsymbol{y}$ are regular in S, and the line segments in S containing $\boldsymbol{x}$ and $\boldsymbol{y}$ are parallel (that is, for all sufficiently small $\boldsymbol{z}$, $\boldsymbol{x} + \boldsymbol{z} \in S \leftrightarrow \boldsymbol{y} + \boldsymbol{z} \in S$).

Let $MinSame(\boldsymbol{x}, t, y)$ say that y is minimal such that $Cong(\boldsymbol{x}, t, y)$, that is, $Cong(\boldsymbol{x}, t, y) \wedge \forall u(Cong(\boldsymbol{x}, t, u) \rightarrow y \leq u)$.

Let $Far(t)$ say that there are no singular points of S with horizontal coordinate $\geq t$. Let $Vert$ be the sentence asserting that S contains a vertical line. Finally, let ϕ be the sentence

$Vert \vee \exists \boldsymbol{x}\ (Reg(\boldsymbol{x}) \wedge \forall t\ \forall t'\ [Far(t) \wedge Far(t') \rightarrow$
$\exists y\ \exists y'[MinSame(\boldsymbol{x}, t, y) \wedge MinSame(\boldsymbol{x}, t', y') \wedge (\boldsymbol{x} + (t', y') - (t, y)) \in S\,]\,]\,).$

We claim that ϕ is the required sentence. Suppose S is a thin semi-linear set in $Cont.Line$. If S contains a vertical line then it satisfies ϕ. Suppose S contains a nonvertical line L, and let $\boldsymbol{x}$ be any regular point on L. Let t and t' be points beyond any singular point of S with $0 < t < t'$. To the right of t, S can only consist of finitely many rays that never intersect, including the part of L beyond t. Among these rays is a lowest ray L' which is parallel to L. There exist y, y' such that (t, y) and (t', y') are on L'. Then $MinSame(\boldsymbol{x}, t, y)$ and $MinSame(\boldsymbol{x}, t', y')$. Since L' is parallel to L, the sum of $\boldsymbol{x}$ and the vector between (t, y) and (t', y') is on L, and hence is in S.

Conversely, suppose that S satisfies ϕ. Suppose $Vert$ does not hold, and let $\boldsymbol{x}$ be any witness for the rest of ϕ. Locally, S looks like a line L through $\boldsymbol{x}$, since $\boldsymbol{x}$ is a regular point and S is thin. We show that L is actually contained in S. Let m be the slope of L. For any d we can find t and t' satisfying $Far(t) \wedge Far(t')$ with $t - t' = d$. Since S satisfies ϕ, there are minimal y and y' such that in neighborhoods of both (t, y) and (t', y'), S is a line with slope m. Since t and t' are far out, the only possibility is that (t, y) and (t', y') actually are on the same

line of S, which means that the slope of the line segment between them must be m. Hence when we add the vector between (t, y) and (t', y') to $\boldsymbol{x}$, we get the point on L at horizontal distance d away from $\boldsymbol{x}$. By assumption, this point is in S. Since d was arbitrary, this shows that S contains L. □

The above argument also shows somewhat more.

Corollary 2. *Cont.Line is FO_{LIN} definable over the family of semi-linear sets with 1-bounded interior.*

5 Definability of n-Linked

We say a semi-linear set A is *n-linked* if for any two points $\boldsymbol{x}$ and $\boldsymbol{y}$ in A, there is a polygonal path from $\boldsymbol{x}$ to $\boldsymbol{y}$ consisting of at most n line segments. n-linkedness is a natural connectivity property of semi-linear sets, and in this section we will give a fairly complete description of the definability of n-linkedness for $n \in N$ (summarized at the end of the section). Somewhat surprisingly, the answer will be "yes" for some values of n and "no" for others.

Theorem 5. *1-linked is FO_{LIN} definable. 2-linked is FO_{LIN} definable over the family of thin semi-linear sets.*

Proof: 1-linked is definable by the statement: For any two points in S, their midpoint is in S.

For 2-linked, let $Parallel(\boldsymbol{x}, \boldsymbol{y})$ say that $\boldsymbol{y}$ is regular and there is a line segment of S containing $\boldsymbol{x}$ that has the same slope as the line segment through $\boldsymbol{y}$. Since S is thin, this can be expressed by the formula:

$$\boldsymbol{x} \in S \wedge Reg(\boldsymbol{y}) \wedge (\forall \boldsymbol{u} \text{ sufficiently close to } \mathbf{0})[\boldsymbol{y}+\boldsymbol{u} \in S \rightarrow (\boldsymbol{x}+\boldsymbol{u} \in S \vee \boldsymbol{x}-\boldsymbol{u} \in S)].$$

As before, we let $Mid(\boldsymbol{x}, \boldsymbol{y})$ denote the midpoint between $\boldsymbol{x}$ and $\boldsymbol{y}$. Now consider the sentence ϕ:

$$\begin{aligned} \forall \boldsymbol{x} \forall \boldsymbol{y}\ (\, Reg(\boldsymbol{x}) \wedge Reg(\boldsymbol{y}) \rightarrow [\, Parallel(\boldsymbol{x}, \boldsymbol{y}) \rightarrow Mid(\boldsymbol{x}, \boldsymbol{y}) \in S\,] \wedge \\ [\, \neg Parallel(\boldsymbol{x}, \boldsymbol{y}) \rightarrow \exists \boldsymbol{z}\ (\, \neg Reg(\boldsymbol{z}) \wedge \\ Parallel(\boldsymbol{z}, \boldsymbol{x}) \wedge Parallel(\boldsymbol{z}, \boldsymbol{y}) \wedge Mid(\boldsymbol{z}, \boldsymbol{y}) \in S \wedge Mid(\boldsymbol{z}, \boldsymbol{x}) \in S\,)\,]\,) \end{aligned}$$

This sentence says that any two regular points either have the same slope and their midpoint is in S, or they have different slopes and there is a singular point realizing both slopes whose midpoint with each of the original points is in S.

We assume ϕ, and show that S is 2-linked. Note that it suffices to show that every two regular points are connected by a path consisting of two line segments (since for two singular points $\boldsymbol{x}$ and $\boldsymbol{y}$ we can find regular points $\boldsymbol{x}'$ and $\boldsymbol{y}'$ nearby such that any path from $\boldsymbol{x}'$ to $\boldsymbol{y}'$ extends to a path from $\boldsymbol{x}$ to $\boldsymbol{y}$ with the same number of segments). Suppose $\boldsymbol{x}$ and $\boldsymbol{y}$ are in S and are regular. Case 1: The

slope of S at $\boldsymbol{x}$ is the same as the slope at $\boldsymbol{y}$. In this case the line segment L between $\boldsymbol{x}$ and $\boldsymbol{y}$ must be in S; if not, we can find regular points $\boldsymbol{p}$ and $\boldsymbol{q}$ on L such that the midpoint of $\boldsymbol{p}$ and $\boldsymbol{q}$ is not in S, which contradicts ϕ. Case 2: The slopes at $\boldsymbol{x}$ and $\boldsymbol{y}$ are different. Choose $\boldsymbol{z}$ as given by ϕ. Reasoning as before, we see that the line segment between $\boldsymbol{z}$ and $\boldsymbol{y}$ must be in S, and the same for the segment between $\boldsymbol{z}$ and $\boldsymbol{x}$. But this shows that S is 2-linked.

Conversely suppose that S is two-linked. For any regular $\boldsymbol{x}$ and $\boldsymbol{y}$ in S, either they are one-linked — in which case they satisfy the $Parallel(\boldsymbol{x}, \boldsymbol{y})$ clause — or they are (minimally) two-linked – in which case they satisfy the $\neg Parallel(\boldsymbol{x}, \boldsymbol{y})$ clause. □

We now show that the assumption that A is thin is necessary:

Corollary 3. *2-linked is not FO_{LIN} definable, even over the family of 1-bounded semi-linear sets with ten singular points.*

Proof: This follows from the proof of Theorem 2. That proof uses a pair of 1-bounded hypersemi-linear structures with ten singular points which are elementarily equivalent in FO_{LIN}, but one structure is 2-linked and the other is not. □

The following results can be proved by modifying the constructions in this paper. The proofs will be given in the full paper.

Theorem 6. *3-linked is not FO_{LIN} definable over the family of thin semi-linear sets.*

Theorem 7. *There is no FO_{LIN} sentence ϕ such that*

$$(4\text{-}linked \rightarrow \phi) \wedge (\phi \rightarrow Conn)$$

holds in all thin semi-linear sets A. In particular, for each $k \geq 4$, k-linked is not FO_{LIN} definable over the family of thin semi-linear sets.

We do not know whether the above theorem can be improved by replacing 4-linked by 3-linked.

Putting together all of the above results we have:

Summary of Definability for n-linked			
n	Semi-linear	... and Thin	... and Boundedly Many Singularities
1	Yes	Yes	Yes
2	No	Yes	Yes
≥ 3	No	No	Yes

Here 'Yes' in a box for integer n and class C means that n-linked is definable over C.

6 Conclusions and Future Work

Questions concerning definability with an extra predicate, even for a well-understood structure such as the real ordered group, turn out to be surprisingly complex. The answers are also a bit counterintuitive: the results here show that seemingly slight modifications of either the query definition or the class of definable sets can make or break definability. It would clearly be desirable to find general topological conditions on a family of sets that guarantee definability and include the interesting definable examples here. Our results, however, indicate that this will be a difficult (perhaps impossible) task.

Given the undefinability results, it seems natural to look for intermediate languages between the first-order linear and polynomial query languages, which can define the queries considered here. In the full version of this paper we will introduce such intermediate languages, but space does not permit us to present the results in this extended abstract.

References

1. F. Afrati, S. Cosmadakis, S. Grumbach and G. Kuper. Linear vs. polynomial constraints in database query languages. In *Proceedings of the Second International Workshop on Principles and Practice of Constraint Programming* pp. 181-192. Springer LNCS 874, 1995.
2. F. Afrati, T. Andronikos,and T. Kavalieros. On the Expressiveness of Query Languages with Linear Constraints; Capturing Desirable Spatial Properties. In *Proceedings of the Second International Workshop on Constraint Database Systems, CDB '97*, pp. 105-115. Springer LNCS, Vol. 1191, 1997.
3. M. Benedikt, G. Dong, L. Libkin and L. Wong. Relational expressive power of constraint query languages. *J. ACM*, 45 (1998), pp. 1–34.
4. M. Benedikt and H. J. Keisler. Expressive Power of Unary Counters In *Structures in Logic and Computer Science*, Mycielski, Rozenberg, and Salomaa (eds.) Springer LNCS 1261 1997.
5. O. Chapuis and P. Koiran. Definability of Geometric properties in algebraically closed fields. To appear in *Mathematical Logic Quarterly.*
6. C.C. Chang and H.J. Keisler. *Model Theory.* North Holland, 1990.
7. P. Kanellakis, G. Kuper, and P. Revesz. Constraint query languages. *JCSS*, 51 (1995), 26–52.
8. B. Kuijpers, J. Paredaens and J. Van den Bussche. On topological elementary equivalence of spatial databases. In *Proceedings of the Sixth International Conference on Database Theory* , pp. 432–446. Springer LNCS 1186, 1997.
9. G. Kuper, L. Libkin and J. Paredaens, eds. *Constraint Databases.* Springer Verlag, 2000.

Bounded Arithmetic and Descriptive Complexity

Achim Blumensath

Mathematische Grundlagen der Informatik
RWTH Aachen, D-52056 Aachen
blume@i7.informatik.rwth-aachen.de

Abstract. We study definability of languages in arithmetic and the free monoid by bounded versions of fixed-point and transitive-closure logics. In particular we give logical characterisations of complexity classes $\mathscr{C}$ by showing that a language belongs to $\mathscr{C}$ if and only if it is definable in either arithmetic or the free monoid by a formula of a certain logic. We investigate in which cases the bounds of fixed-point operators may be omitted. Finally, a general translation of results from descriptive complexity to the approach described in this paper is presented.

Keywords: descriptive complexity, definability, arithmetic

1 Introduction

Descriptive complexity theory studies the connections between definability and complexity classes (see [1,4,5] for an overview). The most common approach originates in finite model theory and yields characterisations of the following form: "Some class $\mathscr{K}$ of finite structures belongs to the complexity class $\mathscr{C}$ if and only if $\mathscr{K}$ is the class of finite models of some sentence of the logic $\mathfrak{L}$." More formally, $\mathscr{K} \in \mathscr{C}$ iff $\mathscr{K} = \operatorname{Mod}(\varphi)$ for some $\varphi \in \mathfrak{L}$. Starting with Fagin's famous characterisation of NPTIME descriptions of most of the common complexity classes have been obtained in this way.

Another equally well developed method is based on function algebras and recursion schemes (see [3] for an overview, or [6] for a formulation in terms of proof theory). It originated in recursion theory with characterisations of the recursive and primitive recursive functions and later on was applied by Cobham to describe the class of polynomial time computable functions.

In the present article we will follow a third approach. We fix a model with universe $\{0,1\}^*$, $\mathbb{N}$, or some other countable set with canonical encoding in $\{0,1\}^*$, and investigate which languages are definable within this model using different logics. This approach has mainly been used in recursion theory so far, for instance to define the arithmetic and analytic hierarchy. To the author's knowledge there are only few characterisations of decidable complexity classes using this method. The Büchi-Bruyère Theorem (see [2] for an overview) states that the p-adic encoding of a set of natural numbers is regular if and only if the set is first-order definable in $(\mathbb{N}, +, V_p)$ where $V_p(x) := p^k$ for the greatest k such that $p^k \mid x$. Wrathall [7] showed that the class of languages definable by Δ_0^0-formulae in

P. Clote and H. Schwichtenberg (Eds.): CSL 2000, LNCS 1862, pp. 232–246, 2000.

$(\mathbb{N}, +, \cdot)$ is equal to the linear hierarchy. As mentioned in [6], a characterisation of the polynomial hierarchy is obtained if one adds the operator $x \# y = 2^{\log_2 x \log_2 y}$.

Below we will show that by adding fixed-point or transitive-closure operators those results can be extended to characterise many of the usual complexity classes. The results themselves are unsurprising and mirror those of finite model theory. Indeed, the similarity between both approaches enables us to present a translation from the formalism of finite model theory to definability in the binary tree and vice versa. So, what are the differences between them? First, our formalism seems to be more general since by using other structures than the binary tree—e.g., arithmetic—we can capture different classes such as EXPTIME or EXPSPACE for which there is no "classical" characterisation. A second point is that depending on the circumstances one approach might be more convenient to work with. For instance, from an an algorithmic point of view the classical approach seems to be more suitable since one can speak about, say, graphs directly instead of having to encode them as words. On the other hand when dealing with languages, e.g., in structural complexity, or when thinking of applications in feasible model theory, our formalism might be of advantage. Finally, by changing the formalism a different set of logical and algebraic methods for the investigation of complexity classes becomes available (although whether this is of any help remains to be seen).

The paper is organised as follows. In the next section we give a short overview of classical descriptive complexity theory and list some results for comparison. Furthermore, we introduce the logics used in the rest of the article.

Section 3 considers various structures of natural numbers and investigates which complexity classes can be characterisations within them. We show how to generalise these results to arbitrary linear orderings of type ω, and study in which cases the bounds of fixed-point operators are really needed.

In Section 4 we turn to the free monoid and show that many classical results can be translated to our approach, and vice versa.

2 Preliminaries

We recall the basic definitions of descriptive complexity theory. For simplicity we will consider only languages over a binary alphabet. In the classical approach each word $w \in \{0,1\}^+$ is represented by the word model

$$\underline{w} := \big(\{0, \dots, |w| - 1\}, <, S, \min, \max, P\big)$$

where S is the successor relation of $<$, min and max are the first and last elements, and P is the set of positions carrying the symbol 1. While in descriptive complexity theory one usually allows classes of arbitrary finite models we will only consider word models in the following.

Let $\mathscr{C}$ be a complexity class. We say that the logic $\mathfrak{L}$ *captures* $\mathscr{C}$ (on word models) iff $\mathscr{C} = \{ L(\varphi) \mid \varphi \in \mathfrak{L} \}$, where $L(\varphi) := \{ w \in \{0,1\} \mid \underline{w} \models \varphi \}$.

Logics capturing complexity classes include first-order logic FO and its extensions by transitive-closure or fixed-point operators, and fragments of second-order logic (see Table 1).

Table 1. Logics capturing complexity classes

Class	Logic
AC^0	FO
Logspace	FO(DTC)
Nlogspace	FO(TC)

Class	Logic
Ptime	FO(LFP)
— " —	Σ^1_1-Horn
— " —	SO-Horn

Class	Logic
Nptime	Σ^1_1
PH	SO
Pspace	FO(PFP)

(Deterministic) transitive-closure logic FO((D)TC) is obtained from FO by adding the operator

$$[(\mathrm{D})\mathrm{TC}_{\bar{x},\bar{y}}\ \varphi(\bar{x},\bar{y},\bar{z})](\bar{u},\bar{v}).$$

The semantics is defined as follows (where for notational convenience we omitted all references to the structure in question). $[\mathrm{TC}_{\bar{x},\bar{y}}\ \varphi](\bar{a},\bar{b})$ holds iff there are tuples $\bar{a}_0 = \bar{a}, \bar{a}_1, \ldots, \bar{a}_n = \bar{b}$, $n > 0$, such that $\varphi(a_i, a_{i+1})$ holds for all $i < n$. The deterministic version is defined by

$$\begin{aligned}&[\mathrm{DTC}_{\bar{x},\bar{y}}\ \varphi(\bar{x},\bar{y},\bar{z})](\bar{u},\bar{v})\\ &\qquad\equiv \big[\mathrm{TC}_{\bar{x},\bar{y}}\ \varphi(\bar{x},\bar{y},\bar{z}) \wedge \forall\bar{y}'(\varphi(\bar{x},\bar{y}',\bar{z}) \to \bar{y}' = \bar{y})\big](\bar{u},\bar{v})\end{aligned}$$

Similarly, in least and partial fixed-point logic FO(LFP) and FO(PFP) one adds the operator

$$[\mathrm{L/PFP}_{R,\bar{x}}\ \varphi(R,\bar{x},\bar{z})](\bar{u})$$

where in the case of LFP, R occurs only positive in φ. To define the semantics consider the operator

$$F(R) := \{\, \bar{a} \mid \varphi(R,\bar{a}) \text{ holds} \,\}.$$

$[\mathrm{LFP}_{R,\bar{x}}\ \varphi](\bar{a})$ holds iff $\bar{a}$ is in the least fixed-point of F, and $[\mathrm{PFP}_{R,\bar{x}}\ \varphi](\bar{a})$ holds iff there is some n such that $F^{n+1}(\emptyset) = F^n(\emptyset)$ and $\bar{a} \in F^n(\emptyset)$.

Finally, denote full second-order logic by SO, existential second-order logic by Σ^1_1, and (existential) second-order horn logic by SO-Horn and Σ^1_1-Horn, respectively. Here, SO-Horn consists of second-order formulae in prenex-normalform where the first-order part is universal, in conjunctive normalform, and each clause contains at most one positive literal $X\bar{x}$ for second-order variables X.

In this article we want to ask which languages can be defined within some fixed structure $\mathfrak{A}$. Of course, in order to do so the universe of $\mathfrak{A}$ should either consists of $\{0,1\}^*$ or we have to choose some encoding of the elements of $\mathfrak{A}$ by words.

Definition 1. *Let $\mathfrak{A}$ be a countable structure and suppose $e : A \to \{0,1\}^*$ is bijective. Let $\mathscr{C}$ be a complexity class. We say that the logic $\mathfrak{L}$* captures *$\mathscr{C}$ on $\mathfrak{A}$ iff*

$$\mathscr{C} = \{\, e(\varphi^{\mathfrak{A}}) \mid \varphi(x) \in \mathfrak{L} \,\},$$

where $\varphi^{\mathfrak{A}} := \{\, a \in A \mid \mathfrak{A} \models \varphi(a) \,\}$.

Obviously this definition may be generalised to relations of arbitrary arity. As there are pairing functions definable in all structures considered below the arity can w.l.o.g. assumed to be one. So, for simplicity, we will only deal with this case.

Below we will investigate which classes are captured on several variants of arithmetic and the free monoid. Since the first-order theory of arithmetic is highly undecidable, we can only hope to capture decidable complexity classes by fragments of FO. In particular we will try to ensure that all variables only range over finite sets. The following definition was motivated by the observation that in recursion theory "bounded quantifiers come for free."

Definition 2. *Fix some structure* $\mathfrak{A}$. *A* bounded guard *on* $\mathfrak{A}$ *is a quantifier-free formula* $\alpha(\bar{x};\bar{y})$ *such that for all* $\bar{b} \in A^m$ *the set* $\{\, \bar{a} \in A^n \mid \mathfrak{A} \models \alpha(\bar{a};\bar{b}) \,\}$ *is finite. Here,* $\bar{x}$ *are called the* bounded variables *of* α, *and* $\bar{y}$ *are the* free variables *or* parameters *of* α.

The bounded fragment BFO *on* $\mathfrak{A}$ *is defined like* FO *where all quantifiers are guarded, i.e., of the form* $(Q\bar{x}.\alpha)\varphi$ *for* $Q \in \{\exists, \forall\}$ *and some bounded guard* α *with bounded variables* $\bar{x}$.

For $O \in \{\mathrm{DTC}, \mathrm{TC}, \mathrm{LFP}, \mathrm{PFP}\}$ *we define the bounded version* BO *by restricting the syntax to*

$$[(\mathrm{D})\mathrm{TC}_{\bar{x},\bar{y}}\ \alpha(\bar{y};\bar{z}) \wedge \varphi(\bar{x},\bar{y},\bar{z})] \quad \textit{and} \quad [\mathrm{L/PFP}_{R,\bar{x}}\ \alpha(\bar{x};\bar{z}) \wedge \varphi(R,\bar{x},\bar{z})]$$

for some bounded guard α. *Let* BFO(O) *be the logic obtained by adding the operator* O *to* BFO. *Similarly, bounded second-order logic* BSO *is obtained by adding (unrestricted) second-order quantifiers to* BFO.

Definition 3. *Let* $\varphi(\bar{x})$ *be a formula of some bounded logic, and let* y *be a variable appearing bound in* φ *(w.l.o.g. assume that no variable is quantified twice). For values* $\bar{c}$ *of* $\bar{x}$, *the* domain of y at $\bar{c}$ *is defined inductively as follows. Let* $(Qy.\alpha(y;\bar{x},\bar{z}))\psi$ *be the subformula where* y *is bound.*

$$\mathrm{dom}(y) := \{\, a \mid \textit{there are } \bar{b} \textit{ in the domains of } \bar{z} \textit{ such that } \alpha(a;\bar{c},\bar{b}) \textit{ holds} \,\}.$$

Intuitively, the domain contains all values y may have. Note that, by induction the domains of bound variables are finite.

Remark 4. Regarding the expressive power the following inclusions hold:

$$\begin{array}{ccccccccc}
\mathrm{FO} & \subseteq & \mathrm{FO(DTC)} & \subseteq & \mathrm{FO(TC)} & \subseteq & \mathrm{FO(LFP)} & \subseteq & \mathrm{FO(PFP)} \\
\cup| & & \cup| & & \cup| & & \cup| & & \cup| \\
\mathrm{BFO} & \subseteq & \mathrm{BFO(BDTC)} & \subseteq & \mathrm{BFO(BTC)} & \subseteq & \mathrm{BFO(BLFP)} & \subseteq & \mathrm{BFO(BPFP)}
\end{array}$$

3 Arithmetic and High Complexity Classes

In this section we will consider $(\mathbb{N}, <, \mathscr{F})$, the natural numbers with order and some additional functions $f \in \mathscr{F}$ where $\mathscr{F}$ is allowed to be empty. Note that in this case we can w.l.o.g. assume that all guards are of the form

$$\bar{x} < t(\bar{y}) \ := \ x_0 < t(\bar{y}) \wedge \cdots \wedge x_{n-1} < t(\bar{y})$$

for some $\mathscr{F}$-term t. The expressive power of bounded logics mainly depends on the growth-rate of the bounds. In order to compare such rates we define $\mathscr{F}_0 \leq \mathscr{F}_1$ for classes $\mathscr{F}_0$ and $\mathscr{F}_1$ of functions on $\mathbb{N}$ iff for all $f_0 \in \mathscr{F}_0$ there is some $f_1 \in \mathscr{F}_1$ such that $f_0(n,\dots,n) \leq f_1(n,\dots,n)$ for all $n \in \mathbb{N}$, and we write $\mathscr{F}_0 \equiv \mathscr{F}_1$ iff both $\mathscr{F}_0 \leq \mathscr{F}_1$ and $\mathscr{F}_1 \leq \mathscr{F}_0$.

Let $\mathcal{T}\mathscr{F}$ be the set of terms built from functions of $\mathscr{F}$. We will see that it mainly depends on the growth-rate of $\mathcal{T}\mathscr{F}$ which complexity classes can be captured on $(\mathbb{N}, <, \mathscr{F})$.

In order to define the complexity of a set of natural numbers it is assumed that numbers are coded by their binary encoding in reversed order, i.e., with the least significant bit first. Note that the number of bits of n is $\lceil \log_2(n+1) \rceil$. Thus, given a monotone function $f : \mathbb{N}^k \to \mathbb{N}$ and a tuple $\bar{n} \in \mathbb{N}^k$ where n_i has l_i bits, the number of bits of $f(\bar{n})$ is

$$(\mathcal{B}f)(\bar{l}) := \big\lceil \log_2\big(f(2^{l_0-1},\dots,2^{l_{k-1}-1})+1\big)\big\rceil.$$

Our first result is preceeded by some two lemmas. Let $f : \mathbb{N}^k \to \mathbb{N}$ be a function such that $f(\bar{a}) \geq a_i$ for all $i < k$. We call a formula $\varphi(\bar{x})$ *f-bounded* iff for all $\bar{a} \in \mathbb{N}^k$ and every term $t(\bar{x},\bar{y})$ in $\varphi(\bar{x})$ the inequality $t(\bar{a},\bar{b}) \leq f(\bar{a})$ holds for values $\bar{b}$ in the domains of $\bar{y}$. Note that for all formulae $\varphi(\bar{x})$ of bounded logics defined above there is some $\mathscr{F}$-term $t(\bar{x})$ such that φ is t-bounded.

Lemma 5. *Let $\mathscr{F}$ be a set of functions whose graphs are decidable in linear space, and let $\varphi(\bar{x}) \in$ BFO be f-bounded. The question whether $(\mathbb{N}, <, \mathscr{F}) \models \varphi(\bar{a})$ can be decided in space $\mathcal{O}(|\varphi| \log_2 f(\bar{a}))$.*

Proof. Since φ is f-bounded we need to consider only values less than $f(\bar{a})$. These can be stored in space $\mathcal{O}(\log_2 f(\bar{a}))$. The claim is proved by induction on φ. To evaluate a function $h(\bar{b})$ we can enumerate all numbers c and check if the tuple $(\bar{b}, c)$ belongs to the graph. Thus, since both the arguments and the values of functions are less than $f(\bar{a})$, atoms can be evaluated in space $\mathcal{O}(\log_2 f(\bar{a}))$. The induction step for boolean connectives is trivial. So consider a formula of the form $(Qy < t(\bar{x}))\psi(y,\bar{x})$. To decide whether it holds we can iterate over all values for y. The only space needed to do so is the storage of y. Thus, it is sufficient to have space $\mathcal{O}(\log_2 f(\bar{a}))$ for each variable appearing in φ. □

The second lemma shows that in many cases we can assume that addition and multiplication is available.

Lemma 6. *The graphs of addition and multiplication are* BFO(BDTC)*-definable in $(\mathbb{N}, <)$.*

Proof. Clearly, 0 and the successor relation S are definable. $+$ and $\cdot$ are defined via the usual recurrence.

$$\begin{aligned} x + y = z :=\ & (y = 0 \wedge x = z) \\ & \vee [\mathrm{DTC}_{uv,u'v'}\ u' < y \wedge v' < z \wedge Su'u \wedge Sv'v](yz, 0x) \\ x \cdot y = z \ :=\ & (y = 0 \wedge z = 0) \\ & \vee [\mathrm{DTC}_{uv,u'v'}\ u' < y \wedge v' < z \wedge Su'u \wedge v' + x = v](yz, 00) \end{aligned}$$

□

The previous lemma indicates that it does not matter much which functions are present since many of them are definable if the logic is at least as expressible as BFO(BDTC). In deed, for such logics, our next result shows that the only thing which matters is the growth-rate of the available functions.

Theorem 7. *Let $\mathscr{R}$ and $\mathscr{F}$ be sets of functions such that $\mathscr{R} \equiv \mathcal{BT}\mathscr{F}$, the graphs of functions in $\mathscr{F}$ are computable in linear space, $\mathcal{O}(n) \subseteq \mathscr{R}$, and $\mathcal{O}\mathscr{R} \subseteq \mathscr{R}$. Let $X \subseteq \mathbb{N}$.*

(i) $X \in \textsc{Dspace}[\mathscr{R}]$ iff X *is* BFO(BDTC)*-definable in* $(\mathbb{N}, \mathscr{F}, <)$.

(ii) $X \in \textsc{Nspace}[\mathscr{R}]$ iff X *is* BFO(BTC)*-definable in* $(\mathbb{N}, \mathscr{F}, <)$.

(iii) $X \in \textsc{Dtime}[2^{\mathscr{R}}]$ iff X *is* BFO(BLFP)*-definable in* $(\mathbb{N}, \mathscr{F}, <)$
iff X *is* $\mathrm{B}\Sigma^1_1$-Horn*-definable in* $(\mathbb{N}, \mathscr{F}, <)$
iff X *is* BSO-Horn*-definable in* $(\mathbb{N}, \mathscr{F}, <)$.

(iv) $X \in \textsc{Ntime}[2^{\mathscr{R}}]$ iff X *is* $\mathrm{B}\Sigma^1_1$*-definable in* $(\mathbb{N}, \mathscr{F}, <)$.

(v) $X \in \textsc{Dspace}[2^{\mathscr{R}}]$ iff X *is* BFO(BPFP)*-definable in* $(\mathbb{N}, \mathscr{F}, <)$.

Proof. In the formulae defined below we will use addition and multiplication whose graphs are definable in all logics mentioned above. In order to keep them readable we will use not only their graphs but also the functions themselves. This can be done since we only use equations of the form $x = t$ for some variable x and term t. Thus all intermediate results are less than or equal to x and we can reduce t by introducing new variables y by bounded quantification $(\exists y \leq x)$.

Below the following model of Turing machine is used. A k-tape Turing machine M is given by a tuple $(Q, \Sigma, \Delta, q_0, F)$ where Q is the set of states, $\Sigma = \{0, 1\}$ is both the input and the working alphabet, q_0 is the initial state, F is the set of final states, and

$$\Delta \subseteq Q \times \Sigma \times \Sigma^k \times \Sigma^k \times Q \times \{-1, 0, 1\}^{k+1}$$

is the transition relation with components: old state, symbol on the input tape, symbols on the working tapes, symbols to write on the working tapes, new state, and movement of the heads.

We prove only two items. The other proofs are similar.

(i) $(\Rightarrow)$ Let $M = (Q, \Sigma, \Delta, q_0, F)$ be an f space-bounded k-tape Turing machine recognising X. W.l.o.g. assume that $Q = \{0, \ldots, n\}$, $\Sigma = \{0, 1\}$, and $q_0 = 0$. Choose some $\mathscr{F}$-term $r(x)$ such that $f(x) \leq \mathcal{B}r(x)$ for all $x \in \mathbb{N}$. Configurations of M can be stored in tuples $(q, \bar{w}, \bar{p})$ where each component is less than $2r(x)$. If there is a formula $\mathrm{TRANS}(\bar{c}, \bar{c}')$ expressing that the configuration stored in $\bar{c}'$ is the successor of $\bar{c}$, we can determine whether a final configuration can be reached from the initial one using an DTC-operator.

$$\varphi_X(x) := (\exists w_1 \cdots w_k p_0 \cdots p_k < 2r(x)) \bigvee_{q_f \in F} [\mathrm{DTC}_{q\bar{w}\bar{p}, q'\bar{w}'\bar{p}'}\ q\text{`}\bar{w}'\bar{p}' < 2r(x) \wedge \mathrm{TRANS}(q\bar{w}\bar{p}, q'\bar{w}'\bar{p}')](0\underbrace{0\cdots0}_{k}\underbrace{1\cdots1}_{k+1}, q_f w_1 \cdots w_k p_0 \cdots p_k).$$

TRANS is defined by

$$\begin{aligned}\mathrm{TRANS}(q\bar{w}\bar{p}, q'\bar{w}'\bar{p}') := \bigvee_{(i,a_0\bar{a},\bar{b},j,\bar{m})\in\Delta} \Big(q = i \wedge q' = j \wedge \mathrm{bit}_{a_0}(x,p_0) \wedge \bigwedge_{l=0}^{k} \mathrm{MOVE}_{m_l}(p_l,p_l') \\ \wedge \bigwedge_{l=1}^{k} (\exists s < w_l)(\exists s' < p_l)\,(w_l = (2s+a_l)p_l + s' \wedge \\ w_l' = (2s+b_l)p_l + s')\Big),\end{aligned}$$

where

$$\begin{aligned}\mathrm{bit}_d(x,p) &:= (\exists s < x)(\exists s' < p)(x = (2s+d)p + s'),\\ \mathrm{MOVE}_m(p,p') &:= \begin{cases} p = 2p' & \text{if } m = -1,\\ p' = p & \text{if } m = 0,\\ p' = 2p & \text{if } m = 1.\end{cases}\end{aligned}$$

($\Leftarrow$) Let X be defined by $\varphi(x)$, and let $\varphi(x)$ be $t(x)$-bounded. Since $\mathscr{R} \equiv \mathcal{BTF}$ there is some $r \in \mathscr{R}$ with $\log_2 t(2^n) \leq r(n)$ for all $n \in \mathbb{N}$. Thus it is sufficient to prove that $X \in \textsc{Dspace}[\mathcal{O}(\log_2 t(2^n))]$. For BFO-formulae this was proved in the above lemma. It remains to consider the evaluation of a DTC-operator $[\mathrm{DTC}_{\bar{x},\bar{y}}\ \bar{z} < s(\bar{z}) \wedge \psi(\bar{x},\bar{y},\bar{z})]$ which can be done by calculating the sequence $\bar{x}_0, \bar{x}_1, \bar{x}_2, \ldots$ of tuples such that $\psi(\bar{x}_i, \bar{x}_{i+1}, \bar{z})$ holds for all i. By induction we can assume that this condition can be checked in $\textsc{Dspace}[\mathcal{O}(\log_2 t(2^n))]$. In order to compute $\bar{x}_{i+1}$ we only need to remember $\bar{x}_i$. Thus the space to store two such tuples is sufficient.

(iii) ($\Rightarrow$) Let $M = (Q, \Sigma, \Delta, q_0, F)$ be an f time-bounded k-tape Turing machine recognising X. W.l.o.g. assume that $Q = \{0, \ldots, n\}$, $\Sigma = \{0,1\}$, and $q_0 = 0$. Choose some $\mathscr{F}$-term $t(x)$ such that $f(x) \leq \mathcal{B}t(x)$ for all $x \in \mathbb{N}$, and let $r(x) = 2t(x) + n$. Using least-fixed points we inductively define relations Q, $\bar{W}$, $\bar{P}$ containing the whole run of M on input x. For instance, $(q,t) \in Q$ means that M is in state q at time t. W.l.o.g. we define those relations by a simultaneous fixed-point which can always be transformed into a normal one.

$$\begin{aligned}\varphi_X(x) := (\exists t < r(x)) \bigvee_{q_f\in F} [\mathrm{LFP}_{Q,qt;\bar{W},apt;\bar{P},pt}\ & qt < r(x) \wedge \psi_Q\\ & apt < r(x) \wedge \psi_{W_1}\\ & \ldots\\ & apt < r(x) \wedge \psi_{W_k}\\ & pt < r(x) \wedge \psi_{P_0}\\ & \ldots\\ & pt < r(x) \wedge \psi_{P_k}]_0(q_f t)\end{aligned}$$

where

$$\begin{aligned}
&\mathrm{CONF}_{q,a_0\bar{a}}(t) := \\
&\quad Qqt \wedge (\exists p < r(x))(\exists s < x)(\exists s' < p)(P_0 pt \wedge x = (2s + a_0)p + s') \\
&\quad \wedge \bigwedge_{l=1}^{k} (\exists p < r(x))(P_l pt \wedge W_l a_l pt) \\
&\psi_Q(q,t) := (q = 0 \wedge t = 0) \vee \bigvee_{(i,a_0\bar{a},\bar{b},j,\bar{m})\in\Delta} (\mathrm{CONF}_{i,a_0\bar{a}}(t-1) \wedge q = j) \\
&\psi_{W_l}(a,p,t) := (a = 0 \wedge t = 0) \\
&\qquad \vee \bigvee_{(i,a_0\bar{a},\bar{b},j,\bar{m})\in\Delta} [\mathrm{CONF}_{i,a_0\bar{a}}(t-1) \wedge \\
&\qquad\qquad (\exists p' < r(x))(P_l p'(t-1) \wedge [(p \neq p' \wedge W_l a p(t-1)) \\
&\qquad\qquad\qquad \vee (p = p' \wedge a = b_l)])] \\
&\psi_{P_l}(p,t) := (p = 1 \wedge t = 0) \\
&\qquad \vee \bigvee_{(i,a_0\bar{a},\bar{b},j,\bar{m})\in\Delta} [\mathrm{CONF}_{i,a_0\bar{a}}(t-1) \wedge \\
&\qquad\qquad (\exists p' < r(x))(P_l p'(t-1) \wedge \mathrm{MOVE}_{m_l}(p,p'))]
\end{aligned}$$

($\Leftarrow$) Let X be defined by $\varphi(x)$, and let $\varphi(x)$ be $t(x)$-bounded. Since $\mathscr{R} \equiv \mathcal{BT}^{\mathscr{F}}$ there is some $r \in \mathscr{R}$ with $\log_2 t(2^n) \leq r(n)$ for all $n \in \mathbb{N}$. Thus, it is sufficient to prove that $X \in \textsc{Dtime}[2^{\mathcal{O}(\log_2 t(2^n))}] = \textsc{Dtime}[\mathcal{O}(t(2^n)^{\mathcal{O}(1)})]$. To evaluate a fixed-point operator $[\mathrm{LFP}_{R,\bar{x}}\ \bar{x} < t(\bar{y}) \wedge \psi(\bar{x},\bar{y})]$ we calculate its stages R^0, R^1, R^2, ... where by boundedness we only need to consider the part $\tilde{R}^i := R^i \cap \{0,\ldots,t(a)-1\}^n$. Thus, $\tilde{R}^{i+1}$ can be computed in $t(a)^n$ steps from $\tilde{R}^i$ each of which takes time $\mathcal{O}(t(a)^{\mathcal{O}(1)})$ (by induction). Since the fixed-point is reached after at most $t(a)^n$ stages we obtain a bound of $\mathcal{O}(t(a)^{\mathcal{O}(1)} \cdot t(a)^n \cdot t(a)^n)$. □

To apply this theorem we need to define functions of appropriate growth. Let $x \# y := 2^{\lceil \log_2 x \rceil \lceil \log_2 y \rceil}$. (Note that $\#$ is associative and commutative.) Since

$$\begin{aligned}
\mathcal{BT}\{+,\cdot\} &\equiv \mathcal{O}(n), \\
\mathcal{BT}\{+,\cdot,\#\} &\equiv \mathcal{O}(n^{\mathcal{O}(1)}), \\
\mathcal{BT}\{+,\cdot,2^n\} &\equiv \mathcal{T}\{2^n\}
\end{aligned}$$

we obtain the results in Table 2. What happens when no functions are present?

Theorem 8. *Let $X \subseteq \mathbb{N}$. The results of the previous theorem also hold for $\mathscr{F} = \emptyset$ and $\mathscr{R} = \mathcal{O}(n)$.*

Proof. The only place where the proofs above fail is the existence of a term $r(x)$ providing a bound large enough to store either the complete contents of a tape or the position of a cell on the tape. For $\mathscr{R} = \mathcal{O}(n)$ this term would be $r(x) := x^c$

Table 2. Logics capturing complexity classes on arithmetic

Class	Logic	Structure
DSPACE$[\mathcal{O}(n)]$	BFO(BDTC)	$(\mathbb{N}, <, +, \cdot)$
NSPACE$[\mathcal{O}(n)]$	BFO(BTC)	$(\mathbb{N}, <, +, \cdot)$
PSPACE	BFO(BTC)	$(\mathbb{N}, <, +, \cdot, \#)$
DTIME$[2^{\mathcal{O}(n)}]$	BFO(BLFP)	$(\mathbb{N}, <, +, \cdot)$
NTIME$[2^{\mathcal{O}(n)}]$	BΣ^1_1	$(\mathbb{N}, <, +, \cdot)$
EXPTIME	BFO(BLFP)	$(\mathbb{N}, <, +, \cdot, \#)$
NEXPTIME	BΣ^1_1	$(\mathbb{N}, <, +, \cdot, \#)$
DSPACE$[2^{\mathcal{O}(n)}]$	BFO(BPFP)	$(\mathbb{N}, <, +, \cdot)$
EXPSPACE	BFO(BPFP)	$(\mathbb{N}, <, +, \cdot, \#)$
ELEMENTARY	BFO(BDTC)	$(\mathbb{N}, <, +, \cdot, 2^n)$

for some c. Though such an r is not available we can handle values of this size by storing each in c variables. Using the (BFO-definable) lexicographic order on c-tuples we can then define addition and multiplication as above. □

The only property of $(\mathbb{N}, <, \mathscr{F})$ used in the proofs above was the order type and the growth-rate of $\mathscr{F}$-terms. This enables us to generalise the results to arbitrary structures as follows. Let $\mathfrak{A} = (A, <, R_0, \dots, R_r, f_0, \dots, f_s)$ be a linearly ordered structure of order type ω. For $a \in A$ let $|a| := \{\, b \in A \mid b < a \,\}$. If we identify elements $a \in A$ by the natural number $|a|$ we get the isomorphic structure $(\mathbb{N}, <, R'_0, \dots, R'_r, f'_0, \dots, f'_s)$ to which we can apply our capturing results. If the complexity of subsets $X \subseteq A$ is measured with regard to the encoding $a \mapsto |a|$ we obtain

Theorem 9. *Let $\mathfrak{A} = (A, <, R_0, \dots, R_r, f_0, \dots, f_s)$ be a linearly ordered structure of order type ω such that R_i, $i \leq r$, and the graphs of f_i, $i \leq s$, are computable in linear space. Let $\mathscr{F} := \{f_0, \dots, f_s\}$ and let $\mathscr{R}$ be a set of functions such that if $\mathscr{F}$ is empty then $\mathscr{R} = \mathcal{O}(n)$, otherwise $\mathscr{R} \equiv \mathcal{BT}\mathscr{F}$, $\mathcal{O}(n) \subseteq \mathscr{R}$, and $\mathcal{O}\mathscr{R} \subseteq \mathscr{R}$. Let $X \subseteq A$.*

(i) $X \in$ DSPACE$[\mathscr{R}]$ iff X *is* BFO(BDTC)*-definable in* $\mathfrak{A}$.

(ii) $X \in$ NSPACE$[\mathscr{R}]$ iff X *is* BFO(BTC)*-definable in* $\mathfrak{A}$.

(iii) $X \in$ DTIME$[2^{\mathscr{R}}]$ iff X *is* BFO(BLFP)*-definable in* $\mathfrak{A}$
iff X *is* BΣ^1_1-HORN*-definable in* $\mathfrak{A}$
iff X *is* BSO-HORN*-definable in* $\mathfrak{A}$.

(iv) $X \in$ NTIME$[2^{\mathscr{R}}]$ iff X *is* BΣ^1_1*-definable in* $\mathfrak{A}$.

(v) $X \in$ DSPACE$[2^{\mathscr{R}}]$ iff X *is* BFO(BPFP)*-definable in* $\mathfrak{A}$.

So far, we only considered extensions of first-order logic. Next we look at the expressive power of BFO. An old result provides an answer in the case of the structure $(\mathbb{N}, <, +, \cdot)$.

Theorem 10 (Wrathall [7]). *X belongs to the linear hierarchy iff X is* BFO*-definable in* $(\mathbb{N}, <, +, \cdot)$.

As mentioned in [6], by adding the operator # characterisation of PH is obtained.

Theorem 11. *$X \in$ PH iff X is* BFO*-definable in* $(\mathbb{N}, <, +, \cdot, \#)$.

Proof. ($\Leftarrow$) Let X be defined by

$$\varphi(x) = (Q_0 y_0 < t_0) \cdots (Q_{n-1} y_{n-1} < t_{n-1}) \psi(x, \bar{y})$$

where ψ is quantifier-free. There is some $k \in \mathbb{N}$ such that $\varphi(x)$ is $(2^{\log_2^k x})$-bounded. Hence each y_i $(i < n)$ can be encoded in $(\log_2 x)^k$ bits. Obviously, quantifier-free formulae $\psi(\bar{a})$ can be evaluated in polynomial time with respect to the length of $\bar{a}$. Thus,

$$X := \{\, x \mid Q_0^{\mathrm{p}} y_0 \cdots Q_{n-1}^{\mathrm{p}} y_{n-1} R(x, \bar{y}) \,\}$$

where all quantifiers are polynomial bounded and

$$R(x, \bar{y}) := y_0 < t_0 \wedge \cdots \wedge y_{n-1} < t_{n-1} \wedge \psi(x, \bar{y})$$

is a PTIME-predicate. Hence, $X \in$ PH.

($\Rightarrow$) By a corollary to Fagin's characterisation of NPTIME, there is some $\varphi \in$ SO such that $x \in X$ iff $\underline{x} \models \varphi$ for all $x \in \{0,1\}^+$ where $\underline{x}$ is the word model of x. We construct a formula $\tilde{\varphi}(x) \in$ BFO with

$$\underline{x} \models \varphi \text{ iff } (\mathbb{N}, <, 0, 1, +, \cdot, \#) \models \tilde{\varphi}(\mathrm{val}(x1))$$

where $\mathrm{val}(y)$ is the number whose binary encoding in reversed order is y. Define

$$\tilde{\varphi}(x) := (\exists p < x + 1)(P_2 p \wedge x < 2p \wedge \varphi^*(x, p))$$

where p denotes the position of the final digit,

$$P_2 x := x = 1 \vee (\forall y < x + 1)(y \mid x \wedge y \neq 1 \rightarrow 2 \mid y)]$$

defines the powers of 2, and φ^* is constructed such that

$$\begin{aligned} &\underline{x} \models \psi(U_0, \ldots, U_{n-1}, y_0, \ldots, y_{m-1}) \\ \text{iff } &(\mathbb{N}, <, 0, 1, +, \cdot, \#) \models \psi^*(x, p, u_0, \ldots, u_{n-1}, 2^{y_0}, \ldots, 2^{y_{m-1}}) \end{aligned}$$

where $u_i := \sum \{\, 2^{l_0 + l_1 |x| + \cdots + l_{k-1} |x|^{k-1}} \mid (l_0, \ldots, l_{k-1}) \in U_i \,\}$. Define

$$\begin{aligned}
(y_0 = y_1)^* &:= y_0 = y_1 \\
(y_0 < y_1)^* &:= y_0 < y_1 \\
(Py)^* &:= \mathrm{bit}(x, y) \\
(U y_0 \ldots y_{k-1})^* &:= \mathrm{bit}(u, y_0 (y_1 \# p) \cdots (y_{k-1} \# p \# \cdots \# p)) \\
(\neg\psi)^* &:= \neg\psi^* \\
(\psi \vee \vartheta)^* &:= \psi^* \vee \vartheta^* \\
(\exists y \psi)^* &:= (\exists y < p)(P_2 y \wedge \psi^*) \\
(\exists U \psi)^* &:= (\exists u < p \# \cdots \# p)\psi^*
\end{aligned}$$

where $\mathrm{bit}(x, y)$ expresses that the bit of x at position y is 1

$$\mathrm{bit}(x, y) := (\exists s < x)(\exists s' < y)(x = (2s + 1)y + s'). \qquad \square$$

Theorem 12. *$X \subseteq \mathbb{N}$ is of elementary complexity iff X is* BFO*-definable in $(\mathbb{N}, <, +, \cdot, 2^n)$.*

The proof is done by directly coding computations of Turing machines. In this case fixed-points are not needed since numbers large enough to code whole runs are available.

Remark 13. The results of Theorems 7 (i), (ii), and 10–12 also hold for oracle machines if one adds the orcale set as unary predicate to the structure.

Unbounded fixed-points. Above we met the boundedness requirement for the logics considered by an ad hoc definition of bounded fixed-points. Next we will investigate under which conditions this can be avoided by using normal (unbounded) operators instead. The first result shows that in many situations it can not.

Proposition 14. *Any relation which is* FO(DTC)*-definable in $(\mathbb{N}, <, +, \cdot)$ (in particular any arithmetic relation) is already* BFO(DTC)*-definable in $(\mathbb{N}, <)$.*

Proof. Since addition and multiplication are FO(DTC)-definable it is sufficient to show how to emulate unbounded quantifiers by DTC-operators. To simulate $\exists x \varphi$ we can enumerate all numbers until some n with $\varphi(n)$ is found. Formally,

$$\exists x \varphi \equiv [\mathrm{DTC}_{x,x'}\ (\neg\varphi(x) \wedge x' = x + 1) \vee (\varphi(x) \wedge x' = 0)](0, 0). \qquad \square$$

In contrast, for purely relational structures a positive result is obtained. Note that the proof above shows that it does not hold for transitive-closure operators.

Proposition 15. *Let $\mathfrak{A} = (A, <, R_0, \ldots, R_m)$ be a relational structure of order type ω, and let $X \subseteq A^k$.*

(i) *X is* BFO(LFP)*-definable if and only if it is* BFO(BLFP)*-definable.*
(ii) *X is* BFO(PFP)*-definable if and only if it is* BFO(BPFP)*-definable.*

Proof. ($\Leftarrow$) is trivial. For ($\Rightarrow$) consider the stages R^0, R^1, ... of the fixed-point induction of $\psi(\bar{y}, \bar{z}) := [\mathrm{LFP}_{R,\bar{x}}\ \varphi(\bar{x}, \bar{y})](\bar{z})$. Since all bounds are of the form $u < v$ for variables u and v the decision whether $\bar{x} \in R^{i+1}$ depends only on values of R^i for arguments less than

$$t := \max\{x_0, \ldots, x_n, y_0, \ldots y_m, z_0, \ldots, z_l\}.$$

In particular, the value at position $\bar{z}$ only depends on lower positions. Therefore we can replace the operator by an bounded one.

$$\psi(\bar{y}, \bar{z}) \equiv \bigvee_i \big(\max(y_i, \bar{y}, \bar{z}) \wedge \chi(y_i)\big) \vee \bigvee_i \big(\max(z_i, \bar{y}, \bar{z}) \wedge \chi(z_i)\big)$$

Table 3. Logics capturing complexity classes on arithmetic

Class	Logic	Structure
LinH	BFO	$(\mathbb{N}, <, +, \cdot)$
PH	BFO	$(\mathbb{N}, <, +, \cdot, \#)$
Dspace$[\mathcal{O}(n)]$	BFO(BDTC)	$(\mathbb{N}, <)$
Nspace$[\mathcal{O}(n)]$	BFO(BTC)	$(\mathbb{N}, <)$
Pspace	BFO(BTC)	$(\mathbb{N}, <, \#)$
Dtime$[2^{\mathcal{O}(n)}]$	BFO(BLFP)	$(\mathbb{N}, <)$
Ntime$[2^{\mathcal{O}(n)}]$	BΣ^1_1	$(\mathbb{N}, <)$
Exptime	BFO(BLFP)	$(\mathbb{N}, <, \#)$
Nexptime	BΣ^1_1	$(\mathbb{N}, <, \#)$
Dspace$[2^{\mathcal{O}(n)}]$	BFO(BPFP)	$(\mathbb{N}, <)$
Expspace	BFO(BPFP)	$(\mathbb{N}, <, \#)$
Elementary	BFO	$(\mathbb{N}, <, +, \cdot, 2^n)$

where $\max(u, \bar{y}, \bar{z}) := \bigwedge_k y_k \leq u \wedge \bigwedge_k z_k \leq u$ says that u is a maximal element, and

$$\chi(u) := [\mathrm{LFP}_{R,\bar{x}}\ x_0 \leq u \wedge \cdots \wedge x_n \leq u \wedge \varphi(\bar{x}, \bar{y})](\bar{z})$$

is the bounded version of the LFP-operator. The proof for BFO(PFP) is identical. □

The characterisations of standard complexity classes we have obtained is summarised in the table above. The results remain valid if we add any relations or functions computable in the respective class. In particular we may add 0, 1, $+$, and $\cdot$. Also the structure $(\mathbb{N}, <)$ may be replaced by any linear order $(A, <)$ of the same order type. Similarly, $(\mathbb{N}, <, \#)$ may be replaced by $(A, <, f)$ where $2^{\log_2^c |a|} \leq |f(a)| \leq 2^{\log_2^d |a|}$ for some c, $d > 1$.

4 The Free Monoid and Low Complexity Classes

So far, we have obtained only characterisations of high (above Ptime) complexity classes. Intuitively, this was caused by the fact that, in arithmetic with the usual order, numbers of n bits have about 2^n predecessors. If we are interested in low complexity classes we thus have to choose a different order. In the classical approach variables can range over n positions in a word model of length n. Therefore, we next consider the free monoid with prefix-ordering where words of length n have n predecessors.

Definition 16. *Let* $\mathfrak{T} := (\{0,1\}^*, \sigma_0, \sigma_1, \prec)$ *where*

$$\sigma_i xy \ :\text{iff}\ \ y = xi, \qquad \textit{and} \qquad x \prec y \ :\text{iff}\ \ y = xz \textit{ for some } z \neq \varepsilon.$$

It turns out that this choice enables us to translate many of the classical results to our setting and vice versa. Let $\mathscr{L}$ consists of the following logics: FO,

FO(DTC), FO(TC), FO(LFP), FO(PFP), Σ^1_1-Horn, Σ^1_1, SO-Horn, and SO. For $\mathfrak{L} \in \mathscr{L}$ denote by $\mathrm{B}\mathfrak{L}$ the corresponding bounded version.

Theorem 17. *Let $X \subseteq \{0,1\}^+$ and $\mathfrak{L} \in \mathscr{L}$. The following statements are equivalent:*

(i) *There is some $\varphi \in \mathfrak{L}$ such that $w \in X$ iff $\underline{w} \models \varphi$.*
(ii) *There is some $\varphi(x) \in \mathrm{B}\mathfrak{L}$ such that $w \in X$ iff $\mathfrak{T} \models \varphi(w)$.*

The familiar results of descriptive complexity theory can thus be stated as

Corollary 18. *Let $X \subseteq \{0,1\}^*$.*

(i) $X \in$ Logspace iff X *is* BFO(BDTC)*-definable in* $\mathfrak{T}$
(ii) $X \in$ Nlogspace iff X *is* BFO(BTC)*-definable in* $\mathfrak{T}$
(iii) $X \in$ Ptime iff X *is* BFO(BLFP)*-definable in* $\mathfrak{T}$
iff X *is* BΣ^1_1-Horn*-definable in* $\mathfrak{T}$
iff X *is* BSO-Horn*-definable in* $\mathfrak{T}$.
(iv) $X \in$ Nptime iff X *is* BΣ^1_1*-definable in* $\mathfrak{T}$
(v) $X \in$ PH iff X *is* BSO*-definable in* $\mathfrak{T}$
(vi) $X \in$ Pspace iff X *is* BFO(BPFP)*-definable in* $\mathfrak{T}$

The proof of Theorem 17 is divided into two propositions.

Lemma 19. *For every $\varphi \in \mathfrak{L}$ there is some $\varphi^*(x) \in \mathrm{B}\mathfrak{L}$ such that, for all $w \in \{0,1\}^+$, $\underline{w} \models \varphi$ iff $\mathfrak{T} \models \varphi^*(w)$.*

Proof. We construct $\varphi^*(x)$ such that for all subformulae ψ the following condition is satisfied:

$$\underline{x} \models \psi(X_0, \dots, X_n, y_0, \dots, y_m) \text{ iff } \mathfrak{T} \models \psi^*(x, X_0^*, \dots, X_n^*, y_0^*, \dots, y_m^*)$$

where y_i^* is the prefix of x of length y_i, and X_i^* contains a tuple of prefixes of x iff the tuple of their lengths is in X_i. In the following definition variables named y, y_0, etc. are bounded, whereas x is the only free variable.

$$\begin{aligned}
(y_0 = y_1)^* &:= y_0 = y_1 & (\neg\psi)^* &:= \neg\psi^* \\
(Py)^* &:= (\exists y' \preceq x)\sigma_1 y y' & (\psi \vee \vartheta)^* &:= \psi^* \vee \vartheta^* \\
(y_0 < y_1)^* &:= y_0 \prec y_1 & (\exists y \psi)^* &:= (\exists y \prec x)\psi^* \\
(X\bar{y})^* &:= X\bar{y} & (\exists X \psi)^* &:= (\exists X)\psi^*
\end{aligned}$$

$$\begin{aligned}
\big([(\mathrm{D})\mathrm{TC}_{\bar{u},\bar{v}}\ \psi](\bar{t},\bar{t}')\big)^* &:= [(\mathrm{D})\mathrm{TC}_{\bar{u},\bar{v}}\ v_0 \prec x \wedge \dots \wedge v_{k-1} \prec x \wedge \psi^*](\bar{t},\bar{t}') \\
\big([X\mathrm{FP}_{R,\bar{u}}\ \psi](\bar{t})\big)^* &:= [X\mathrm{FP}_{R,\bar{u}}\ \bar{u} \prec x \wedge \dots \wedge u_{k-1} \prec x \wedge \psi^*](\bar{t})
\end{aligned}$$

□

Lemma 20. *For every $\varphi(x) \in \mathrm{B}\mathfrak{L}$ there is some $\varphi' \in \mathfrak{L}$ such that, for all $w \in \{0,1\}^+$, $\mathfrak{T} \models \varphi(w)$ iff $\underline{w} \models \varphi'$.*

Proof. Note that, since $\varphi(x)$ has only one free variable x, all bounded variables are prefixes of x. Thus, it again is sufficient to ensure that

$$\mathfrak{T} \models \psi(x, X_0, \dots, X_n, y_0, \dots, y_m) \text{ iff } \underline{x} \models \psi'(X'_0, \dots, X'_n, |y_0|, \dots, |y_m|)$$

where the definition of X'_i is slightly more involved since there are $|x|+1$ prefixes of x, but only $|x|$ elements in $\underline{x}$. Therefore, we double the arity of X_i and define

$$X'_i := \{\, (u'_{00}u'_{01}, \dots, u'_{k0}u'_{k1}) \mid (u_0, \dots, u_k) \in X_i \,\}$$

where

$$(u'_{j0}, u'_{j1}) := \begin{cases} (0, u_j) & \text{if } u_j \prec x \\ (|x|-1, |x|-1) & \text{if } u_j = x. \end{cases}$$

In the following definition variables named y, y_0, etc. are bounded, whereas x is the only free variable. t stands for an arbitrary term which can either be on of y, x, or one of y, min, max. Furthermore, $P^i t$ is Pt for $i = 1$, and $\neg Pt$ for $i = 0$.

$$\begin{array}{llll}
(y_0 = y_1)' & := y_0 = y_1 & (\sigma_i y_0 y_1)' & := Sy_0y_1 \wedge P^i y_0 \\
(y = x)' & := \text{false} & (\sigma_i x t)' & := \text{false} \\
(x = x)' & := \text{true} & (\sigma_i y x)' & := y = \max \wedge P^i y \\
(y_0 \prec y_1)' & := y_0 < y_1 & (y \prec x)' & := \text{true} \\
(x \prec t)' & := \text{false} & & \\
(\neg\psi)' & := \neg\psi' & ((\exists y_0 \prec y_1)\psi)' & := \exists y_0(y_0 < y_1 \wedge \psi') \\
(\psi \vee \vartheta)' & := \psi' \vee \vartheta' & ((\exists y \prec x)\psi)' & := \exists y \psi' \\
(\exists X^k \psi)' & := \exists X^{2k} \psi' & (X t_0 \dots t_{k-1})' & := X \tilde{t}_0 \dots \tilde{t}_{k-1}
\end{array}$$

where

$$\tilde{t} := \begin{cases} \min y & \text{if } t = y, \\ \max \max & \text{if } t = x. \end{cases}$$

$$\big([(\mathrm{D})\mathrm{TC}_{\bar{u},\bar{v}}\ v_0 \prec t_0 \wedge \dots \wedge v_{n-1} \prec t_{n-1} \wedge \psi](\bar{t}', \bar{t}'')\big)' :=$$
$$[(\mathrm{D})\mathrm{TC}_{\bar{u},\bar{v}}\ v_0 < t_0 \wedge \dots \wedge v_{n-1} < t_{n-1} \wedge \psi'](\bar{t}', \bar{t}'')$$
$$\big([X\mathrm{FP}_{R,\bar{u}}\ u_0 \prec t_0 \wedge \dots \wedge u_{n-1} \prec t_{n-1} \wedge \psi](\bar{t}')\big)' :=$$
$$[X\mathrm{FP}_{R,\bar{u}}\ u_0 < t_0 \wedge \dots \wedge u_{n-1} < t_{n-1} \wedge \psi'](\bar{t}') \qquad \square$$

Remark 21. (i) The preceding results can be generalised to formulae with several free variables.

(ii) Nothing changes if we replace σ_0, σ_1 by the corresponding functions or even add concatenation. For the last part note that for all variables y appearing in a formula $\varphi(x)$ there is some k such that y ranges over values of the form $y_0 \cdots y_j$, $j < k$, where the y_i are prefixes of x. Hence the value of each y can be

Table 4. Logics capturing complexity classes on the free monoid

Class	Logic	Structure
AC^0	BFO	$(\mathfrak{T}, \mathrm{bit})$
LOGSPACE	BFO(BDTC)	$\mathfrak{T}$
NLOGSPACE	BFO(BTC)	$\mathfrak{T}$
PTIME	BFO(LFP)	$\mathfrak{T}$
NPTIME	$B\Sigma^1_1$	$\mathfrak{T}$
PH	BSO	$\mathfrak{T}$
PSPACE	BFO(PFP)	$\mathfrak{T}$

stored in a fixed number of variables and we can eliminate concatenation by its BFO(BDTC)-definition.

(iii) Since $\mathfrak{T}$ is relational bounded LFP- and PFP-operators can be replaced by unbounded ones as in the case of arithmetic.

(iv) If one adds to $\mathfrak{T}$ either the relations $|x| + |y| = |z|$ and $|x| \cdot |y| = |z|$, or the relation $\mathrm{bit}(x, y)$ saying that the $|y|^{\mathrm{th}}$ bit of $|x|$ is 1, and considers word models with analogous predicates, we also can characterise the class AC^0, i.e., $X \subseteq \{0,1\}^*$ is in AC^0 iff X is BFO-definable in $(\mathfrak{T}, \mathrm{bit})$.

References

1. S. ABITEBOUL, R. HULL, AND V. VIANU, *Foundations of Databases*, Addison-Wesley, 1995.
2. V. BRUYÈRE, G. HANSEL, C. MICHAUX, AND R. VILLEMAIRE, *Logic and p-recognizable sets of integers*, Bull. Belg. Math. Soc., 1 (1994), pp. 191–238.
3. P. CLOTE, *Computation models and function algebras*, in Handbook of Computability Theory, E. R. Griffor, ed., North-Holland, 1999.
4. H.-D. EBBINGHAUS AND J. FLUM, *Finite Model Theory*, Springer, 1995.
5. N. IMMERMAN, *Descriptive Complexity*, Springer, New York, 1998.
6. J. KRAJÍČEK, *Bounded Arithmetic, Propositional Logic, and Complexity Theory*, Cambridge University Press, 1995.
7. C. WRATHALL, *Rudimentary predicates and relative computation*, SIAM Journal on Computing, 7 (1978), pp. 194–209.

Independence: Logics and Concurrency

J.C. Bradfield

Laboratory for Foundations of Computer Science, Division of Informatics,
King's Bldgs, University of Edinburgh, Edinburgh EH9 3JZ, UK.
jcb@dcs.ed.ac.uk

Abstract. We consider Hintikka et al.'s 'independence-friendly first-order logic'. We apply it to a modal logic setting, defining a notion of 'independent' modal logic, and we examine the associated fixpoint logics.

1 Introduction

Modal and temporal logics have a long history as system specification languages in computer science, and computer scientists' study of temporal logic has generated many interesting theoretical developments, as well as many important practical advances. The logics in question typically describe properties of execution paths of systems, either explicitly, as in LTL and CTL, or via a 'next-step' operator and fixpoints, as in the modal mu-calculus.

Another major issue in computer science is concurrency. The theoretical and practical analysis of concurrent and distributed systems also has a long history. Probably the most successful approach to understanding concurrency at a fundamental semantic level is the use of 'independence models', in which certain events are stated to be independent of other events, and the associated partial order semantics, in which a partial order of causality is established between events. There have also been several major practical advances in exploiting partial order structure, such as stubborn sets and sleep sets [22] and unfolding techniques [16].

Naturally, one wishes to have modal and temporal logics for concurrent systems. There are several ways to apply the paradigm of normal temporal logic, typically by working on event structures, or similar models, and having explicit operators representing concurrency or independence: see [19] for a survey of such logics. In some models, and therefore in the associated logics, the notion of independence is abstract; in others, it is more concrete, perhaps derived from a notion of location; but in all cases it inheres in the model rather than the logic.

However, there is also a notion of inherent independence in logic, which seems quite natural when specifying concurrent systems, but which has not, with one notable exception, received much attention in concurrency theory.

In 1961, Leon Henkin [11] introduced the quantifier $\begin{smallmatrix}\forall x & \exists y \\ \forall u & \exists v\end{smallmatrix}$, which is intended to mean that the choice of y depends only on x, *not* on u; and similarly the choice of v depends only on u; a formal semantics is given by using suitable Skolem functions.

The Henkin quantifier and its generalizations received some attention: the main papers were at the beginning of the 70s, by William Walkoe [23] and

P. Clote and H. Schwichtenberg (Eds.): CSL 2000, LNCS 1862, pp. 247–261, 2000.

Herbert Enderton [9], and later also by Krynicki and Barwise. It was shown that $\begin{smallmatrix}\forall & \exists \\ \vee & \exists\end{smallmatrix}$ suffices to express all *partially ordered* or *branching* quantifiers (that is, quantifiers where the dependence of variables need not be simply linear). More recently, Georg Gottlob has looked at generalized quantifiers, including $\begin{smallmatrix}\forall & \exists \\ \forall & \exists\end{smallmatrix}$, for capturing complexity classes.

In the last few years, logicians, in particular the (Boston and) Helsinki logicians Jaakko Hintikka, Gabriel Sandu and Jouko Väänänen, have returned with a vengeance to the study of 'independent' quantifiers as a basic logic, rather than a specialist extension, and Hintikka and Sandu [12] have gone so far as to claim that their absence from Frege's logic was a 'fundamental error', a 'horror', and to claim that their reintroduction heralds 'a revolution in logic' (albeit with a question mark in the title). It is not necessary to accept this thesis to accept that 'independence-friendly first-order logic' (IF-FOL) is an interesting, and natural [3,20] logic. For example, when considering the development of systems by several designers working independently, the notion of independent choice seems natural, and may reasonably form part of any language for discussing properties of such designs. At a more concrete level, different components of a distributed system cannot be assumed to have full knowledge of other components, and so cannot make fully informed choices.

In this paper, I suggest that the use of modalities based on Henkin quantifiers gives an approach to independence in concurrency which complements the model-based independence usually used. I start by outlining the work on independent quantifiers; I then give a natural modal logic using independent quantification, and relate it to other work on logics for distributed systems; I define the obvious fixpoint extension, and consider complexity issues and the extent to which the usual theory transfers. I then return to the first-order independence logic giving our metalanguage, and consider briefly the interpretation of fixpoints there: which is not a trivial question. Finally, I raise the problem of more general modal fixpoint logics based on independence. Owing to space constraints, proofs are generally omitted or just sketched, unless they are of particular interest.

2 'Independence-Friendly' First-Order Logic

By FOL^+ we mean first-order logic *in positive form*: that is, negation is applied only to atomic formulae, and both operators of the dual pairs $\vee/\wedge$ and $\exists/\forall$ are considered primitive; FOL denotes the usual logic where negation is not so restricted. $\mathrm{FOL}^+{+}Q$ denotes the logic where the additional operator Q appears only positively. We use similar notation for other logics.

2.1 Partial Quantifiers *à la* Henkin

A *branching quantifier* Q is a set $\{x_1, \ldots, x_m, y_1, \ldots, y_n\}$ of variables, carrying a partial order $\prec$; the x_i are universal, the y_i existential. The semantics of $Q\phi$ is defined to be that of $\exists f_1 \ldots f_n . \forall x_1 \ldots x_m . \phi[f_i(y_i\downarrow)/y_i]$, where $y_i\downarrow$ is the list

of variables $\prec y_i$, and $[\cdot/\cdot]$ denotes syntactic substitution: thus f_i is a Skolem function for y_i, but it refers only to variables preceding y_i in the partial order.

In particular, the *Henkin quantifier* $\begin{smallmatrix}\forall & \exists \\ \forall & \exists\end{smallmatrix} = \{x_1, x_2, y_1, y_2\}$ with $x_i \prec y_i$ is written $\begin{smallmatrix}\forall x_1 & \exists y_1 \\ \forall x_2 & \exists y_2\end{smallmatrix}$; thus $\begin{smallmatrix}\forall x & \exists y \\ \forall u & \exists v\end{smallmatrix}\phi(x, y, u, v)$ is equivalent by definition to $\exists f, g. \forall x, u.\, \phi(x, f(x), u, g(u))$.

Fact 1. The following are the basic properties of branching quantifiers.

1. Any formula $Q\phi$ where Q is a branching quantifier and ϕ is FOL$^+$ is equivalent to a formula of FOL$^+ + \begin{smallmatrix}\forall & \exists \\ \forall & \exists\end{smallmatrix}$. [23,9].
2. By definition, any FOL$^+ + \begin{smallmatrix}\forall & \exists \\ \forall & \exists\end{smallmatrix}$ formula is equivalent to an existential second-order formula.
3. Moreover, any existential second-order formula is equivalent to a formula of FOL$^+ + \begin{smallmatrix}\forall & \exists \\ \forall & \exists\end{smallmatrix}$. (This is because the formula can be 'unskolemized' using $\begin{smallmatrix}\forall & \exists \\ \forall & \exists\end{smallmatrix}$: for example, the assertion that there exists an injective function which never takes a particular value c, which being true only on infinite structures is not first-order, can be written as $\exists f. \forall x, u.\, (f(x) = f(u) \Rightarrow x = u) \wedge (f(x) \neq c)$; and this can be written using $\begin{smallmatrix}\forall & \exists \\ \forall & \exists\end{smallmatrix}$ as $\begin{smallmatrix}\forall x & \exists y \\ \forall u & \exists v\end{smallmatrix}.(x = u \Rightarrow y = v) \wedge (y = v \Rightarrow x = u) \wedge (y \neq c)$.)
4. It follows via Fagin's Theorem that on finite structures FOL$^+ + \begin{smallmatrix}\forall & \exists \\ \forall & \exists\end{smallmatrix}$ expresses NP-hard properties [4]; in fact, as one would expect, FOL$+\begin{smallmatrix}\forall & \exists \\ \forall & \exists\end{smallmatrix}$ captures L$^{\mathrm{NP}}$ [10].

2.2 Partial Knowledge Games and Quantifiers *à la* Hintikka

An alternative way of giving semantics to branching quantifiers is via games. Recall the Hintikka model-checking game for FOL$^+$: given a formula ψ and a structure M, a position is a subformula $\phi(\boldsymbol{x})$ of ψ together with a *deal* for ϕ, that is, an assignment of values $\boldsymbol{v}$ to its free variables $\boldsymbol{x}$. At a position $(\forall x.\, \phi_1, \boldsymbol{v})$, Abelard chooses a value v for x, and play moves to the position $(\phi_1, \boldsymbol{v} \cdot v)$; similarly Eloise moves at $\exists x.\, \phi$. At $\phi_1 \wedge \phi_2$, Abelard chooses a conjunct; and at $\phi_1 \vee \phi_2$, Eloise chooses a disjunct. A play of the game terminates at (negated) atoms $P(\boldsymbol{x})$ (resp. $\neg P(\boldsymbol{x})$), and is won by Eloise (resp. Abelard) iff $P(\boldsymbol{x})$ is true with the current deal. Then it is standard that $M \vDash \phi$ exactly if Eloise has a winning strategy in this game, where a strategy is a function from sequences of legal positions to moves.

These games have *perfect information*; both players know everything that has happened, and in particular when one player makes a choice, they know the other player's previous choices. Game semantics for the Henkin quantifiers, following [12], use games of imperfect information: in the game for $\begin{smallmatrix}\forall x & \exists y \\ \forall u & \exists v\end{smallmatrix}\phi$, when Eloise chooses for v, she does not know what Abelard chose for x. To make this explicit, the logic is written with a more general syntax which is linear rather than two dimensional. We here use Hodges' [13] syntax, which addresses certain flaws in Hintikka's; and for reasons of space and simplicity, we omit some operators in this version.

IF-FOL$^+$ is obtained from FOL$^+$ by modifying the syntax of quantifiers to be $\forall x/W.$ and $\exists x/W.$, where W is a set of variables. The intention is that W is the set of independent variables, whose values the player is not allowed to know at this choice point. Thus the Henkin quantifier $\begin{smallmatrix}\forall x & \exists y \\ \forall u & \exists v\end{smallmatrix}$ can be written as $\forall x/\varnothing.\, \exists y/\varnothing.\, \forall u/\{x,y\}.\, \exists v/\{x,y\}$. Henceforth we freely omit set braces and write just $\forall x.$ for $\forall x/\varnothing$. We shall also, again for reasons of space and simplicity, consider only the independent $\exists$, leaving the dualization for independent $\forall$ to the reader (or see [13]). The adapted model-checking game, where the information is restricted, can be shown to characterize the Skolem function semantics in the sense that Eloise has a winning strategy iff the formula is true. However, these games are not determined, so it is *not* true that Abelard has a winning strategy iff the formula is false. For example, $\begin{smallmatrix}\forall x \\ \exists y\end{smallmatrix} x = y$ (or $\forall x.\, \exists y/x.\, x = y$) is false in any structure with more than one element, but Abelard has no winning strategy.

3 Henkin Modal Logic

For a concurrency theorist, it is natural to see the model-checking game for $\begin{smallmatrix}\forall x & \exists y \\ \forall u & \exists v\end{smallmatrix}\phi$ not as a game of imperfect information, but as a *concurrent* game: at $\begin{smallmatrix}\forall x & \exists y \\ \forall u & \exists v\end{smallmatrix}\phi$ the game splits into two independent concurrent components: in one component play proceeds with $\forall x.\, \exists y.$, in the other with $\forall u.\, \exists v.$, and then the components join to proceed with ϕ. This also seems a natural statement to want to make about concurrent systems: a choice in one component should not even be able to depend on a concurrent choice in another! However, until recently all modal or temporal logics enforced logical dependence of choices.

Example 2. In the children's game Scissors–Paper–Stone, the two players (say Abe and Elly) each put one hand behind their back, and make it either open (Paper), a fist (Stone), or a V-sign (Scissors). The two players then *simultaneously* bring forward their hands. The round is won according to the rules: Scissors cut Paper, Paper wraps Stone, Stone blunts Scissors (if both players choose the same object, the round is drawn). Can we ask the question 'can Elly always win?'?

If we formalize the game by seeing Abe as choosing between the three actions $A = \{sc_A, pa_A, st_A\}$, and similarly for Elly, and viewing the game as their independent concurrent composition with final states that satisfy *Ewins* when Elly wins, we can approximate the question by $[A]\langle E\rangle \mathit{Ewins} \wedge \langle E\rangle [A] \mathit{Ewins}$. This expresses that Elly can win on all interleavings; it is (correctly) false, but it is false for the wrong reason: that $\langle E\rangle [A] \mathit{Ewins}$ fails, means that after Elly chooses, Abe can, with knowledge of that choice, make her lose, which mis-models the situation.

If we formalize the simultaneity by using a synchronous concurrent composition, there is, in normal modal or temporal logics, no way to ask the question at all, since only one action (the simultaneous choice) happens.

Of course, with an eye on Henkin quantifiers, the obvious answer is to define a modality $\begin{smallmatrix}[] \\ \langle\rangle\end{smallmatrix}$ so one can write $\begin{smallmatrix}[A] \\ \langle E\rangle\end{smallmatrix} \mathit{Ewins}$, with the intended meaning.

Recently, Alur, Henzinger and Kupferman [1] have implicitly taken this approach in their Alternating Temporal Logic, apparently without awareness of Henkin quantifiers: they express, rather, their logic directly in terms of games and strategies. We shall now define a simple logic of Henkin modalities; a fragment of this provides a generalization of ATL, and also includes several other distributed logics.

3.1 A Distributed System Model

First let us define a notation for system models that, although by no means the most general possible, is sufficient for all our examples. Given an algebra Act of basic actions, sequential components P are built from non-deterministic choice $P_1 + P_2$, action prefix $a.P_1$, and mutual recursive definitions $P = Q(P)$. Systems are defined as a parallel composition $\|_i^{\mathcal{S}} P_i$ of n components; $Act_\perp = Act \cup \{\perp\}$ where $\perp$ is the 'non-action' or idle action, and $\mathcal{S} \subseteq Act_\perp^n \times Act$ gives the synchronization rules by $\|_i^{\mathcal{S}} P_i \xrightarrow{a} \|_i^{\mathcal{S}} P_i'$ (where $a \in Act$) iff each $P_i \xrightarrow{\alpha_i} P_i'$ (for $\alpha_i \in Act_\perp$) and also $\mathcal{S}(\alpha_1, \ldots, \alpha_n, a)$. We also write $\|_i^{\mathcal{S}} P_i \xrightarrow{\bigotimes_i \alpha_i} \|_i^{\mathcal{S}} P_i'$, particularly in the case when $\mathcal{S}$ is a function $Act_\perp^n \to Act$.

This model gives a convenient notation for various distributed automata formalisms, or for finite state CCS or CSP (by adjusting the synchronization); and a slightly less convenient notation for arbitrary finite 1-safe Petri nets, by taking each place as a two state component and using $\mathcal{S}$ to code the transitions. The alternating transition systems of [1] can also be coded into this notation.

3.2 Henkin Modalities

We now define Henkin modal logic (${}^{\mathrm{H}}_{\mathrm{M}}\mathrm{L}$) on such a system thus: in addition to ML^+, that is, basic modal logic in positive form, we have the *concurrent modalities* of the form $\bigotimes_{i=1,\ldots,n} \mathsf{Q}_i^1(A_i^1) \ldots \mathsf{Q}_i^m(A_i^m)$, where $\mathsf{Q}_i^j(A_i^j)$ is either $[A_i^j]$ or $\langle A_i^j \rangle$, and $A_i^j \subseteq Act_\perp$; the *length* of the modality is m. We shall sometimes use '$-$' to mean 'Act' in modalities. These modalities are given a semantics in terms of the corresponding first-order Henkin quantifier. This is notationally tedious to write out, so we just give the semantics of the length 2 Henkin modality $\left[\!\!\left[\begin{smallmatrix} \cdot & \cdot \\ \cdot & \cdot \end{smallmatrix} \right]\!\!\right]$ on a 2-component system, by way of example. (We write $\left[\!\!\left[\begin{smallmatrix} [A] & \langle B \rangle \\ [C] & \langle D \rangle \end{smallmatrix} \right]\!\!\right]$ in Henkin style for the formal $([A]\langle B\rangle \otimes [C]\langle D\rangle)$.)

$$P_1 \|^{\mathcal{S}} P_2 \vDash \left[\!\!\left[\begin{smallmatrix} [A_1] & \langle B_1 \rangle \\ [A_2] & \langle B_2 \rangle \end{smallmatrix} \right]\!\!\right] \phi$$

holds iff

$$\begin{array}{l} \forall \alpha_1 \in A_1, P_1' \, \exists \beta_1 \in B_1, P_1'' \\ \forall \alpha_2 \in A_2, P_2' \, \exists \beta_2 \in B_2, P_2'' \end{array} (P_1 \|^{\mathcal{S}} P_2) \xrightarrow{\alpha_1 \otimes \alpha_2} (P_1' \|^{\mathcal{S}} P_2') \xrightarrow{\beta_1 \otimes \beta_2} (P_1'' \|^{\mathcal{S}} P_2'')$$

Note that by definition ${}^{\mathrm{H}}_{\mathrm{M}}\mathrm{L}$ does not include the duals of the Henkin modalities.

Let $\overset{\text{H}}{\text{M}}\text{L}_n$ denote the sublogic where the modalities have length at most n. $\overset{\text{H}}{\text{M}}\text{L}_1$ with the addition of fixpoints (see later) includes many existing distributed modal logics.

Example 3. The indexed transition systems of Andersen [2] take the form of systems $\|^{\mathcal{S}}_{1\leq i\leq n} P_i$, where $\mathcal{S}(\alpha_1,\ldots,\alpha_n,a)$ iff there is some k such that $\alpha_k = a$ and all other α_i are $\perp$. (The systems move one component at a time.) Then in Andersen's 'polyadic mu-calculus', the indexed modality $[a]_k$ is in our notation $\bigotimes_i [\alpha_i]$ where $\alpha_k = a$ and the other α_i are all $\perp$.

Example 4. The distributed net systems used by Huhn, Niebert and Wallner [14] are systems $\|^{\mathcal{S}}_{1\leq i\leq n} P_i$ where each P_i is a sequential Petri net, and $\mathcal{S}$ defines the synchronization of common transitions. The transitions are labelled, which we may express either by using instead sets of transitions, or by letting $\mathcal{S}$ map local transitions to global labels rather than to global transitions. Their logic is defined locally via an event structure style semantics, so that the basic modality is $\langle a\rangle_J \phi$, where $J \subseteq \{1,\ldots,n\}$; this is true at a *local state* of P_i, for $i \in J$, if a (which must share some location with J) can fire as an immediate action of the J processes, possibly with non-J actions happening first, and then ϕ holds.

Such a modality is not expressible in $\overset{\text{H}}{\text{M}}\text{L}_1$ without fixpoints, since we have an interleaving, global semantics, but with fixpoints it is just $\mu Z.\langle a\rangle\phi \vee (\bigotimes_i \langle A_i\rangle)Z$, where A_i is $\perp$ for $i \in J$ and $Act_\perp$ otherwise.

The previous examples are merely using the idea of location; to use the power of independent quantification, we need:

Example 5. The alternating transition systems of [1] are an unlabelled setting. An ATS is a global state space Q, shared by n agents P_i. Each agent has a transition function $\delta_i : Q \to 2^{2^Q}$. At a state q, each agent chooses $Q_i \in \delta_i(q)$. It is required that for all such choices, $\bigcap_i Q_i$ is a singleton set $\{q'\}$, and this determines the next state of the system. Thus the intuition is that each agent chooses its desired successors, independently of the others, and the system moves to the one state they all desire.

To encode this in our setting, it is easiest to make the choice an explicit local transition happening before the move to the next state. So we take each agent P_i to be a transition system with state space $Q \cup 2^Q$, with transitions $q \xrightarrow{\tau} Q_i$ (where τ is a dummy label) for every $Q_i \in \delta_i(q)$, and $Q_i \xrightarrow{q'} q'$ for every $q' \in Q_i$ (note that q' is also being used as a label). The synchronization algebra $\mathcal{S}$ is then the diagonal on $Q \cup \{\tau\}$, and one move of the original ATS corresponds to two moves of our system $\|^{\mathcal{S}}_i P_i$. Thus the reachable states of our system have the form $(q,q,\ldots,q)$ or $(Q_1,\ldots,Q_n)$, and the requirements on the ATS transition function mean that there is exactly one transition from any reachable $(Q_1,\ldots,Q_n)$, to the unique $(q',q',\ldots,q')$ with $q' \in \bigcap_i Q_i$.

The Alternating Temporal Logic of [1] is like CTL, except that the path quantifiers are not just $\forall$ and $\exists$, but have the form $\langle\langle E\rangle\rangle$, where $E \subseteq \{1,\ldots,n\}$. The interpretation is that, given a path formula ϕ (for example, $\mathbf{F}\psi$, 'eventually

ψ'), $\langle\langle E\rangle\rangle\phi$ is true if the agents E can choose their successors to make ϕ hold on all paths, regardless of how the other agents $\overline{E}$ choose their successors. In other words, the E agents can win the path game defined by ϕ.

This logic is then in turn defined as a fragment of a suitable mu-calculus, just as CTL is defined in the usual modal mu-calculus. The [1] mu-calculus has the basic modality $\langle\langle E\rangle\rangle \bigcirc \phi$, where $\bigcirc$ is the 'next' operator, interpreted as just described.

Now in our encoding, this modality is just $(\bigotimes_i \mathsf{Q}_i)\langle - \rangle \phi$, where Q_i is $\langle\tau\rangle$ for $i \in E$, and $[\tau]$ otherwise.

Remark 6. A referee suggests that my presentation of ${}^{\mathrm{H}}_{\mathrm{M}}\mathrm{L}$ on distributed systems obscures that fact that it is a natural fragment of IF-FOL just as ML is a natural fragment of FOL. There is some force in this, but there is a counter: if one defines the modalities without reference to locality, one must use a model such as ATS's in which locality is encoded by means of sets of possible next states for each agent. It is arguable that the ATS model is less natural than the usual models, and also that the distributed modalities are themselves more natural for the user that a pure Henkin quantifier. A fuller discussion of this point requires further analysis of guarded and finite-variable fragments of IF-FOL, which, as far as I know, has not yet been done.

3.3 Adding Fixpoints

We have already used fixpoint notation above; to justify this, it suffices to note the following fact:

Proposition 7. *A Henkin modality defines a monotone operator on sets of global states. Therefore least and greatest fixpoint operators can be added to* ${}^{H}_{M}L$ *as in the normal modal mu-calculus: call this* $\mu\,{}^{H}_{M}L$.

Because we are defining interpretations as sets of global states, some of the usual theory carries through trivially: for example,

Proposition 8. *The usual simple complexity upper bound applies: given a* $\mu\,{}^{H}_{M}L$ *formula* ϕ *of length* m *and fixpoint alternation depth* d*, and a system of size* n *(meaning here the number of global states plus the number of transitions), the complexity of determing whether* $s \vDash \phi$ *is* $O(m \cdot H \cdot n^d)$*, where* H *is the cost of evaluating a basic modality or boolean.*

In normal modal mu-calculus, H is $O(n)$; we consider below what H is for Henkin modalities.

Other parts of the theory can be made to carry through in a rather uninteresting way, by ignoring the concurrency and using rather the Skolem semantics:

Proposition 9. *Consider the tableau model-checking system of [21], and add rules for the Henkin modalities in the following form:*

$$\frac{s_1 \| s_2 \vdash ([A_1]\langle B_1\rangle \otimes [A_2]\langle B_2\rangle)\phi}{s_{11} \| s_{21} \vdash \phi \ldots s_{1m} \| s_{2m} \vdash \phi}$$

where the states on the bottom are given by: for every A_1 successor s_1' of s_1 there is some B_1 successor s_{1j} of s_1', and similarly for the second component, such that $s_1 \| s_2 \overset{A_1 \otimes A_2}{\longrightarrow} s_1' \| s_2' \overset{B_1 \otimes B_2}{\longrightarrow} s_{1j} \| s_{2j}$.

Then the resulting tableau system is sound and complete.

Other parts do not carry through so easily: for example, the relationships with automata and parity games, where one must extend the usual frameworks with either second-order moves or Henkin quantification in the winning conditions.

3.4 Cost of Henkin Modalities

As we remarked above, the usual complexity analysis carries through, but depends, of course, on the cost of evaluating the modalities.

Proposition 10. *The cost of evaluating a Henkin modality of length 1 is $O(n^3)$.*

Proof. To check $s_1 \| s_2 \in [\![([A] \otimes \langle B \rangle)\phi]\!]$, it suffices to try in turn each of the $O(n)$ possible B-successors of s_2 and check whether all the so chosen $A \otimes B$-successors are included in $[\![\phi]\!]$. □

The result of this is that $\mu^{\mathrm{H}}_{\mathrm{M}} \mathrm{L}_1$ is not significantly worse (in theory!) than normal modal mu-calculus. However, since it is also a slight generalization of the AMC of [1], the complexity hardness results there can also be applied to $\mu^{\mathrm{H}}_{\mathrm{M}} \mathrm{L}_1$, giving, for example, P-hardness (rather than the NL-hardness of modal mu-calculus).

When we move to real Henkin modalities, things become more expensive. As we know that first-order Henkin modalities are NP-complete, we might expect this; but we might also hope that the restricted quantification involved in modal logic reduces the complexity. Unfortunately, this is not the case:

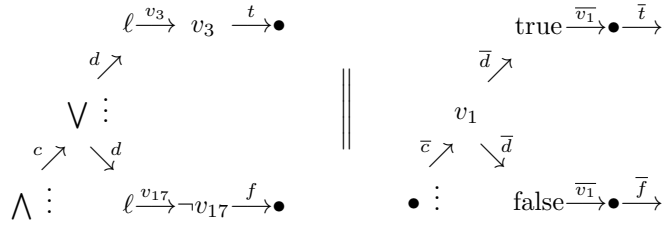
Proposition 11. *Model-checking ${}^{H}_{M}L_2$ is NP-hard (and obviously NP).*

Proof. There is a direct reduction (with thanks to Perdita Stevens) from CNF-SAT, which illustrates the use of the Henkin modalities quite nicely, and so is worth giving in full.

Consider an instance Φ of CNF-SAT: it has the form $\bigwedge_{1 \le i \le m} C_i$, where $C_i = \bigvee_{1 \le j \le n_i} \ell_{ij}$, and each ℓ_{ij} is v_k or $\neg v_k$ for one of the r variables v_k.

We now define a system with two components: the first represents the formula, the second a choice of variable assignment. Let $D = \sum_{1 \le i \le m} c.C_i$, and $C_i = \sum_{1 \le j \le n_i} d.L_{ij}$, and $L_{ij} = v_k.t$ if $\ell_{ij} = v_k$, or $L_{ij} = v_k.f$ if $\ell_{ij} = \neg v_k$

Let $A = \sum_{1 \le k \le r} \overline{c}.V_k$, and $V_k = \overline{d}.\overline{v_k}.\overline{t} + \overline{d}.\overline{v_k}.\overline{f}$.



Now let $\mathcal{S}$ be the synchronization algebra given by $\alpha \otimes \overline{\alpha} = \tau$, and symmetrically, for every $\alpha \in \{c, d, v_k, t, f\}$, and let $P = D \| ^{\mathcal{S}} A$. (In standard CCS notation, this would be $(D \mid A) \backslash \{c, d, v_k, t, f\}$.) Note that the size of this system is at most quadratic in the size of Φ.

Now we claim that $P \vDash \left[\begin{smallmatrix}-\\-\end{smallmatrix}\right]\left\langle\begin{smallmatrix}-\\-\end{smallmatrix}\right\rangle[\tau]\langle\tau\rangle\mathrm{tt}$ iff Φ is satisfiable.

Firstly, suppose Φ has a satisfying assignment S. In the game for $\left[\begin{smallmatrix}-\\-\end{smallmatrix}\right]\left\langle\begin{smallmatrix}-\\-\end{smallmatrix}\right\rangle$, Eloise plays the following strategy: in the top half, in response to Abelard's choice of C_i, she chooses any L_{ij} made true by S – such a literal exists, by definition of satisfying assignment. In the bottom half, in response to Abelard's choice of V_k, she chooses $\overline{v_k}.\overline{t}$ or $\overline{v_k}.\overline{f}$ according as v_k is true or false in S. The resulting process is then $v_{k'}.(t/f) \| ^{\mathcal{S}} \overline{v_k}.(\overline{t}/\overline{f})$ according as ℓ_{ij} is $v_{k'}$ or $\neg v_{k'}$, and as v_k is true or false. If $k' \neq k$, then the process is deadlocked, and so satisfies $[\tau]\langle\tau\rangle\mathrm{tt}$. If $k' = k$, then since Eloise chose L_{ij} to be true in S, the left hand process has t iff the right hand process has $\overline{t}$, and so again the process satisfies $[\tau]\langle\tau\rangle\mathrm{tt}$.

Conversely, if $P \vDash \left[\begin{smallmatrix}-\\-\end{smallmatrix}\right]\left\langle\begin{smallmatrix}-\\-\end{smallmatrix}\right\rangle[\tau]\langle\tau\rangle\mathrm{tt}$, then Eloise's strategy for $\left[\begin{smallmatrix}-\\-\end{smallmatrix}\right]\left\langle\begin{smallmatrix}-\\-\end{smallmatrix}\right\rangle$ defines, in the bottom half, an assignment S, and in the top a selection of one literal L_{ij_i} for each conjunct C_i, such that the literal is made true by the assignment; so S is a satisfying assignment. □

So it appears that the Henkin modality is exponential to check. This has two apparent consequences: it means that one might reasonably argue it is useless, even if there are natural properties to be expressed with it; and it means that the fixpoint alternation depth is no longer the dominant factor in the combined complexity of model-checking.

The first consequence is in any case rather dubious – worst-case complexity may or may not have any relevance to practical complexity, as demonstrated by the practical utility of the theoretically non-elementary Mona system – but both consequences can be mitigated if systems have certain structure.

Given a system in our framework, define the local size to be the maximum of the sizes of the individual components.

Proposition 12. *In a k-component system with local size d, evaluating a Henkin modality of length 2 costs $O(d^{d+2k}) = O(2^{(d+2k)\log d})$.*

Proof. Brute force exploration of all the possible 'Skolem functions': in each component there are at most d^d possible local strategies in the local $[]\langle\rangle$ game; to check that a candidate k-tuple of local strategies satisfies the formula costs at most $d^k \cdot d^k$. Thus the total cost is d^{d+2k} □

In a loosely coupled system, the global state space is of size d^k; thus as the number of components increases, the exponential of the Henkin modality is absorbed by the exponential of the state space explosion, rather than adding to it.

The fact that the Henkin modality in the worst case dominates fixpoint complexity raises the question of the fixpoint alternation hierarchy for finite models. In the *infinite* case, this question can be solved by extending previous techniques.

Proposition 13. *For $\mu^{H}_{M}L_2$ on infinite models (and even on 2-component systems), the fixpoint alternation hierarchy is strict.*

Proof. If one considers arithmetic with fixpoints and the Henkin quantifier, the fixpoint alternation hierarchy is strict – see the next section. Then a natural extension of the techniques of [5], in which a Henkin modality is used to encode a first-order Henkin quantifier, transfers this hierarchy to $\mu^{\mathrm{H}}_{\mathrm{M}}\mathrm{L}_2$ on 2-component systems. □

However, for this result to transfer down to finite models, we would need the finite model property for $\mu^{\mathrm{H}}_{\mathrm{M}}\mathrm{L}_2$ *and* its closure under negation. One can see that:

Proposition 14. *Given a finite action set, fixed number of components, and given synchronization algebra, then if a $\mu^{H}_{M}L_2$ formula has a countably infinite model, it has a finite model.*

Proof. The elegant proofs of the modal mu-calculus finite model property do not easily transfer. However, the brute force construction of a finite model by surgery on the unravelling, does transfer: essentially one builds an infinite tableau (using the tableau rule given in Proposition 9 for the Henkin modality); then removes all branches that are not required to exist by a diamond modality; and then looks for repetitions of the same set of subformulae annotating a state (after defining a suitable notion of subformula for the Henkin modality). Closing repetitions to form loops gives a finite (albeit large) model. □

Unfortunately, I cannot see how to obtain this result for the dual of the Henkin modality, if indeed it holds.

4 IF-FOL and Fixpoints

4.1 Fixpoints and the Henkin Quantifier

We return now to the general setting of first-order logic. As we remarked in the last proposition, there is no great difficulty in combining a simple Henkin quantifier with fixpoints: $\begin{smallmatrix}\forall & \exists \\ \forall & \exists\end{smallmatrix}$ is just another monotone operator. Getting a handle on the expressive power is less trivial. Consider $\mathrm{LFP}^{+}+\begin{smallmatrix}\forall & \exists \\ \forall & \exists\end{smallmatrix}$, that is, first-order logic with fixpoints in positive form, plus the Henkin quantifier occurring only positively, and consider the structure of arithmetic. The Henkin quantifier itself has Σ^1_1 power, so a formula of the form $\mu(z,Z).\begin{smallmatrix}\forall x & \exists y \\ \forall u & \exists v\end{smallmatrix}\phi$, for first-order ϕ, is at worst a fixpoint over Σ^1_1, which is at worst Σ^μ_2 in the normal fixpoint hierarchy (because $\Sigma^1_1 = \Pi^\mu_1$: Kleene's theorem). However, it is not immediately obvious that, say, $\begin{smallmatrix}\forall x & \exists y \\ \forall u & \exists v\end{smallmatrix}\mu(z,Z).\phi$ is well-behaved; naively, it might be Σ^1_2. Of course, it is not, because the fixpoint is parametrized on the variables x, y, u, v, and cannot actually refer to the defined Skolem functions. The key to analysing the expressive power is to extend the normal form results of [15,5] with the

Lemma 15. *In arithmetic (or other structures with suitable coding power), a Henkin quantifier can be pushed inside a fixpoint operator.*

As corollaries of the normal form and existing theorems [6,7] on the power of fixpoints, one obtains

Corollary 16. *The fixpoint alternation hierarchy for* $LFP^+ +^{\forall\,\exists}_{\forall\,\exists}$ *(on arithmetic) is strict.*

Corollary 17. $LFP^+ +^{\forall\,\exists}_{\forall\,\exists}$ *is no more expressive than* LFP^+ *(on arithmetic).*

and in fact both these apply to $\mathrm{LFP}+^{\forall\,\exists}_{\forall\,\exists}$ as well; adding the dual is not problematic. The latter result contrasts sharply with the position on finite models, where a single $^{\forall\,\exists}_{\forall\,\exists}$ is stronger than fixpoints, unless P = NP.

4.2 Fixpoints and Full IF-FOL

Now consider the full independence-friendly logic, with the linear syntax allowing arbitrary specification of independence in quantifiers. We described the Skolem function semantics, and the game semantics of Hintikka. These semantics have the problem that they are not compositional, and in particular they give no meaning to a formula such as $\exists y/x.\,\phi$, which occurs as a subformula of $\forall x.\,\exists y/x.\,\phi$ (alias $^{\forall x}_{\exists y}\phi$). Hintikka and Sandu [12] thought that this was an inevitable fact, and Hintikka even went so far as to say that 'no perverse ingenuity' could produce a compositional semantics. So Wilfrid Hodges [13] promptly gave a compositional semantics. This allows us to add fixpoints freely in the usual way (call the result IF-LFP, 'independence-friendly least fixpoint logic'), but at an (apparently) considerable price: the interpretation of a formula with free variables is no longer just a set of value tuples, but a set of sets of tuples. For the formal semantics, recall the definition of the syntax in section 2.2 – which is, for simplicity, a fragment of the full logic. We will give, using Hodges' terminology, the semantics for this fragment.

Let $\phi(\boldsymbol{x})$ mean that $\boldsymbol{x}$ is all the free variables of ϕ without repetition. Given a structure A, a *deal* for ϕ is an assignment of an element of A to each variable in $\boldsymbol{x}$. The interpretation of a formula is the set of its *trumps*, defined as follows.

- If $P(\boldsymbol{x})$ is atomic, then a non-empty set X of deals is a trump iff every deal in X satisfies P.
- X is a trump for $(\phi \wedge \psi)(\boldsymbol{x})$ iff X is a trump for $\phi(\boldsymbol{x})$ and X is a trump for $\psi(\boldsymbol{x})$.
- X is a trump for $(\phi \vee \psi)(\boldsymbol{x})$ iff it is non-empty and there are trumps U of ϕ and V of ψ such that every deal in X belongs either to U or V.
- X is a trump for $\forall y.\,\psi(\boldsymbol{x}, y)$ iff the set $\{\, \boldsymbol{a}b \mid \boldsymbol{a} \in X, b \in A \,\}$ is a trump for ψ.
- The interesting case is the existential quantifier (and, in the presence of negation, the universal quantifier). Given $\phi(\boldsymbol{x}) = \exists y/W.\,\psi(\boldsymbol{x}, y)$ (where W is a subset of the variables $\boldsymbol{x}$), say that two deals are W-equivalent iff they

agree on all variables not in W. Say that a non-empty set X of deals is a W-set if its members are pairwise W-equivalent. Then a set X of deals for ϕ is a trump iff: there is a trump U for ψ such that for every W-set $Y \subseteq X$ there is a b such that $\{ \boldsymbol{a}b \mid \boldsymbol{a} \in Y \} \subseteq U$.

A trump for ϕ is essentially a set of winning positions for the model-checking game for ϕ, for a given *uniform* strategy, that is, a strategy where choices are uniform in the 'hidden' variables. The meaning $[\![\phi]\!]$ of a formula is then defined to be the set of its trumps.

It is easy to see that any subset of a trump is a trump. In the case of an ordinary first-order $\phi(\boldsymbol{x})$, the set of trumps of ϕ is just the power set of the set of tuples satisfying ϕ. To see how a more complex set of trumps emerges, consider the following formula, which has x free: $\exists y/\{x\}.\, x = y$. Any singleton set of deals is a trump, but no other set of deals is a trump. Thus we obtain that $\forall x.\, \exists y/\{x\}.\, x = y$ has no trumps (unless the domain has only one element).

Now consider adding fixpoints. In normal LFP, we form an inductive definition via a formula $\phi(x, X)$ with a relation variable X. In Hodges' semantics, the 'relation' variable should instead range over sets of (potential) trumps, and hence the fixpoint is taken over functionals $\wp(\wp(A)) \to \wp(\wp(A))$ rather than $\wp(A) \to \wp(A)$. For this, we require

Lemma 18. *The operators $\vee$, $\wedge$, $\forall x.$, $\exists x/W.$ are monotone in the lattice of trump sets.*

Thus given a n-ary 'relation' variable X, and a formula $\phi(\boldsymbol{x}, X)$ with n free variables, we have in the usual way an operator on $\wp(\wp(A^n))$, and we can form the fixpoints $\mu X.\phi$ and $\nu X.\phi$. Note, however, that this is *not* the form of a general inductive definition over $\wp(A^n)$, since we do not have variables ranging directly over $\wp(A^n)$.

The usual machinery of ordinal approximants applies. However, as the fixpoints are over $\wp(\wp(A))$, the naive bound on the closure ordinal of a single fixpoint is now exponential in $|A|$. If A is countably infinite, this raises the possibility of $2^{\aleph_0}$-step approximation; and high expressive power.

Indeed, the question of the expressive power of this logic, on the structure of arithmetic, is most interesting. We know that ordinary induction on the integers (i.e. LFP) is a small fragment of Δ^1_2; but with IF-LFP, we are performing induction over the continuum, which is, unrestricted, extremely powerful: ${}^{\omega}\omega$ induction over a Π^1_1 formula gives Σ^1_2, and then induction over Δ^1_2 gives all the semihyperprojective sets [8] (including, for example, the entire analytical hierarchy). In our framework, it is natural to conjecture that the expressive power of IF-LFP is much less than this (and in fact I conjecture it is in Δ^1_2), but at present I do not have the tools to analyse the power. The reasons for the conjectured weakness are that the form of induction is very restricted, as noted above.

We can, however, note the simple fact that this semantics agrees with the usual semantics for LFP when there is no independence:

Proposition 19. *If an IF-LFP formula ϕ contains no slashes, then its denotation $[\![\phi]\!]^{\mathrm{IF-LFP}}$ is exactly the set of non-empty subsets of the ordinary LFP denotation $[\![\phi]\!]^{\mathrm{LFP}}$; and in particular, $\boldsymbol{x} \in [\![\phi]\!]^{\mathrm{LFP}} \Leftrightarrow \{\boldsymbol{x}\} \in [\![\phi]\!]^{\mathrm{IF-LFP}}$.*

Proof. By induction on approximants and the structure of ϕ. □

5 Independence-Friendly Modal Mu-Calculus?

The fixpoint extension of IF-FOL is interesting in its own right, but it is also a tool for understanding more sophisticated Henkin modal logics than we have defined so far.

We have already used the notation $\otimes$ to describe the concurrent, independent combination of sequences of modalities. A natural extension is to take it as an operator on formulae instead, so one can write, for example, $([a](P_1 \wedge (\langle b\rangle * \wedge\langle c\rangle *))) \otimes ([\overline{a}](P_2 \wedge \langle -\rangle *))Q$, with the intended interpretation that the first component satisfies the first factor (with P_1 probably, but not necessarily, being a *local* proposition as in [14]), making its existential choices independently; the second satisfies the second; and in the process, the two components synchronize on the a; and when they rendezvous at $*$, the system satisfies Q.

The simplest way to define such operators formally is to use a game semantics, in the style of Hintikka. Precisely, take the syntax of modal logic, in positive form, and add: an n-ary (for each n) operator $\otimes$ ("parallel composition"), an atomic formula $*$ ("end of parallel"), and a binary operator . (roughly, "sequential composition"). In our usual framework, only one level of $\otimes$ nesting is allowed, and $\otimes$ and $*$ may only occur on the left side of a "sequential composition".

Given a system (say 2-component, for simplicity) in our usual framework, a formula defines a game thus: the normal modalities are played as usual. At a formula $(\phi_1 \otimes \phi_2).\psi$, the game splits into two concurrent parts: every modality move in one half must match a modality move in the other half, with the global system advancing as dictated by the synchronization algebra, but the choices in one half are independent of those in the other. When play reaches $*$ in both halves, play continues at ψ; if play stops because an atomic formula (other than $*$) is reached in either component, Eloise wins if the formula is locally satisfied (in both halves, in the event that both halves reach an internal atomic formula).

This game semantics, although not unnatural, is quite complex. It is an exercise in applying [13] to see that:

Remark 20. Given appropriate predicates on systems in our framework, the above logic can be given a semantics using IF-FOL, with the IF-FOL game semantics corresponding naturally to the above modal game semantics.

The issue of fixpoints then arises. Provided that fixpoint operators are restricted to 'system formulae', that is, are not allowed inside $\otimes$, there is no particular problem, and we obtain a logic that can express moderately complex properties concerning independence and synchronization, but which is still decidable.

However, it is natural to ask whether formula such as $((\mu Z.*\vee(P_1\wedge[-]Z))\otimes(\mu Z.*\vee(P_2\wedge[-]Z)).\langle a\rangle\mathrm{tt}$ make sense. They appear to make good intuitive sense, when one thinks of μ as just meaning 'finite looping', but it is not apparent how to give a semantics to them within IF-LFP. Furthermore, an attempt to give a game semantics results in games where the two agents can proceed arbitrarily far without 'rendezvous'ing and acquiring knowledge of each other's state space. In general, such games, in which the lack of knowledge persists indefinitely, have undecidable outcome problems; for example, in [1], a general game quantifier temporal logic is defined, which is undecidable on finite systems for that reason. Whether such formulae can be allowed with restrictions sufficient to retain decidability, is an issue for further investigation.

6 Summary and Future Work

In this paper, we have introduced the notion of 'independence-friendly' modal logic; we have laid some of the groundwork for a modal theory, and looked at the extension of the first-order theory by fixpoints. This opens up many avenues for further exploration, both in a 'computer science' setting and in a more 'logical' setting. For example:

- Does full $\mu^{\mathrm{H}}_{\mathrm{M}}\mathrm{L}$ with negation have a finite model property?
- What is the expressive power of IF-LFP on arithmetic?
- Does $\mu^{\mathrm{H}}_{\mathrm{M}}\mathrm{L}$ have a fixpoint alternation hierarchy on finite models? If so, how does this connect to the complexity?
- In this paper, we have used only a fragment of IF-FOL; in particular, we have not used independent disjunction (and conjunction), which is a subtle connective (see [13]); and we have not addressed the issue of negation and duality. The latter question was traditionally ignored, by sticking to positive form; Hodges has provided an account of negation, and there is some work to do in applying it in our setting.
- What is the relation between the *logical* independence we have been studying, and the *model* independence in semantic accounts of true concurrency? There appear to be some links between logics of causality and locality, including history-sensitive logics such as the hereditary history-preserving logic of [17], and these should be investigated.
- Independence-friendly logics are philosophically and mathematically interesting, but are they really useful? Putative applications such as distributed system design should be investigated, and it is hoped to pursue this in a future project.

Finally, I should like to thank the several colleagues with whom I have discussed the idea of $^{\mathrm{H}}_{\mathrm{M}}\mathrm{L}$, in particular Perdita Stevens and Juliana Küster Filipe. I also thank the referees for perceptive criticisms, which, I regret, are largely unanswered here; a longer version of this paper will be found via my home page `http://www.dcs.ed.ac.uk/home/jcb/`. I am supported by EPSRC Advanced Fellowship AF/97/0322.

References

1. R. Alur, T. Henzinger and O. Kupferman, Alternating-time temporal logic, in *Proc. 38th FOCS (1997)*, 100–109.
2. H. R. Andersen, *Verification of Temporal Properties of Concurrent Systems*, DAIMI PB – 445, Computer Science Dept, Aarhus University, 1993.
3. J. Barwise, On branching quantifiers in English, *J. Philos. Logic* **8** 47–80 (1979).
4. A. Blass and Y. Gurevich. Henkin quantifiers and complete problems. *Ann. Pure Appl. Logic* **32**:1, 1–16 (1986).
5. J. C. Bradfield, The modal mu-calculus alternation hierarchy is strict, *Theor. Comput. Sci.* **195** 133–153 (1997).
6. J. C. Bradfield, Fixpoint alternation and the game quantifier, *Proc. CSL '99*, LNCS **1683** 350–361 (1999).
7. J. C. Bradfield, Fixpoint alternation: arithmetic, transition systems, and the binary tree, *Theoret. Informatics Appl.* **33** 341–356 (1999).
8. D. Cenzer, Monotone inductive definitions over the continuum, *J. Symbolic Logic* **41**:1 188–198 (1976).
9. H. B. Enderton, Finite partially ordered quantifiers, *Z. für Math. Logik u. Grundl. Math.* **16** 393–397 (1970).
10. G. Gottlob, Relativized logspace and generalized quantifiers over finite ordered structures. *J. Symbolic Logic* **62**:2, 545–574 (1997).
11. L. Henkin, Some remarks on infinitely long formulas, *Infinitistic Methods*, Pergamon Press, Oxford and PAN, Warsaw, 167–183 (1961).
12. J. Hintikka and G. Sandu, A revolution in logic?, *Nordic J. Philos. Logic* **1**(2) 169–183 (1996).
13. W. Hodges, Compositional semantics for a language of imperfect information, *Int. J. IGPL* **5**(4), 539–563.
14. M. Huhn, P. Niebert and F. Wallner, Verification on local states, *Proc. TACAS '98*, LNCS **1384** 36–51.
15. R. S. Lubarsky, μ-definable sets of integers, *J. Symbolic Logic* **58** (1993) 291–313.
16. K. L. McMillan, Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits, *Proc. CAV'92*, 161–171 (1992).
17. M. Nielsen and C. Clausen, Bisimulations, games and logic. BRICS report RS-94-6. BRICS, Aarhus (1994).
18. W. Penczek, On undecidability of propositional temporal logics on trace systems, *Inf. Proc. Let.* **43**(3) 147–153 (1992).
19. W. Penczek and R. Kuiper, Traces and Logic, in: Diekert and Rozenberg, ed. *The Book of Traces*, World Scientific (1995).
20. G. Sandu, On the logic of information independence and its applications, *J. Philos. Logic* **22** 361-372 (1993).
21. C. P. Stirling and D. J. Walker, A general tableau technique for verifying temporal properties of concurrent programs. *Proc. Int. BCS–FACS Workshop on Semantics for Concurrency* Workshops in Computing (Springer–Verlag, Berlin, 1990) (1–15).
22. A. Valmari, A stubborn attack on state explosion. *Proc. CAV '90*, DIMACS vol. 3, 25–42 (1991).
23. W. J. Walkoe, Jr, Finite partially-ordered quantification. *J. Symbolic Logic* **35** 535–555 (1970).

Flatness Is Not a Weakness

Hubert Comon and Vèronique Cortier

LSV, Ecole Normale Supèrieure de Cachan
61 Avenue du prèsident Wilson, 94235 Cachan cedex, France.
{comon,cortier}@lsv.ens-cachan.fr
Tel: +33 1 47 40 24 30
Fax: +33 1 47 40 24 64

Abstract. We propose an extension, called $\mathcal{L}_p^+$, of the temporal logic LTL, which enables talking about finitely many register values: the models are infinite words over tuples of integers (resp. real numbers). The formulas of $\mathcal{L}_p^+$ are *flat*: on the left of an until, only atomic formulas or LTL formulas are allowed. We prove, in the spirit of the correspondence between automata and temporal logics, that the models of a $\mathcal{L}_p^+$ formula are recognized by a piecewise *flat* counter machine; for each state q, at most one loop of the machine on q may modify the register values.
Emptiness of (piecewise) flat counter machines is decidable (this follows from a result in [9]). It follows that satisfiability and model-checking the negation of a formula are decidable for $\mathcal{L}_p^+$. On the other hand, we show that inclusion is undecidable for such languages. This shows that validity and model-checking positive formulas are undecidable.

Keywords: Counter automata, temporal logics, model-checking, verification, logic in computer science.

1 Introduction

Temporal logics play a central role in the specification and verification of reactive systems (see e.g. [16]). Temporal logics come in two varieties: linear time and branching time [13]. We consider here the linear version PLTL. This (propositional) temporal logic is decidable (actually PSPACE-complete [18]). Model checking is also PSPACE-complete (linear w.r.t. the model). The set of words which satisfy a PLTL formula is recognized by a finite Büchi automaton, which shows the relatively weak expressive power of the logic; here we are interested in specifying and verifying infinite state systems.

One more general (hence more realistic) class of models would be machines with finitely many registers (or counters) taking their values in integers or real numbers and a finite control, of which the simplest example is Minsky machines. Unfortunately, even the most simple temporal property, reachability, is undecidable for 2-counters machines [17]. Several restrictions of this model have been studied. For instance Petri nets basically consist in removing the ability to test a counter for zero. Temporal properties of Petri nets have been studied in, e.g., [15].

P. Clote and H. Schwichtenberg (Eds.): CSL 2000, LNCS 1862, pp. 262–276, 2000.

Another approach consists in adding hypotheses on the control instead of hypotheses on the basic operations only. That is the approach of [9]; a counter machine is called *flat* if there is at most one loop on each state. For such machines, the binary reachability relation between two control states is expressible in Presburger arithmetic [9], hence decidable. Flat automata are still a significant subclass of counter automata since, for instance, Alur and Dill's *timed automata* [2] can be encoded in this model [11].

The notion of flatness appears in several places. As we have seen, it appears to be a crucial hypothesis for counter machines. In [7] the authors study the set of reachable configurations for an automaton communicating through fifo channels. They show how to describe such a set of configurations using a Presburger formula, *provided that the control is flat*. Similarly, in [1] the authors study automata communicating through lossy fifo channels and introduce the so-called SRE which assume a flatness hypothesis on the control. This is not by chance that a similar hypothesis appears in several places: roughly, if only increments are allowed, using one loop one may compute addition and using two nested loops one can compute multiplication; from one loop to two nested loops we move from decidable to undecidable theories.

More interestingly, flatness appears naturally in PLTL itself: following the automata approach, the models of a PLTL formula are recognized by a *weak alternating automaton* (see e.g. [19]). Weakness means that there is an ordering on the states such that any state occurring in the image of q by the transition function is smaller (or equal to) q. Hence "weak" is a synonym of "flat" in the context of alternating automata, though the Büchi automaton accepting the same language as a weak alternating automaton may contain several loops on the same state, hence is not itself flat.

This raises the following question: assume that we design a temporal logic which includes as atomic formulas expressions involving finitely many counters and that we are able to construct for each formula ϕ an automaton which recognizes the models of ϕ, would the automaton be flat ? If this were the case, we could design decision procedures for such a logic, because we do have decision procedures for flat automata.

That is the purpose of the present paper: we define a *flat temporal logic* $\mathcal{L}_p$ whose atomic formulas include expressions such as $x \geq y - 1$ for instance where x, y are integer variables. "flatness" is a restriction in which only atomic formulas may occur on the left of an "until". If we drop such a restriction, we show that we immediately cross the boarder: the logic becomes undecidable. In [4,6,5] there are similar hypotheses: they design a logic in which it is possible to consider as a first class object the number of times a given propositional formula is satisfied. This logic is in general undecidable, but becomes decidable when on the left (resp. on the right) of an until only propositional formulas are allowed. Strictly speaking, the results of [4] for instance are incomparable with ours since neither the logic nor the models we consider are the same. (Roughly, they consider models which are described by a process algebra, i.e. in which there is no explicit counter. On the other hand, the integer variables in the logic only

count number of occurrences of a given event). Let us emphasize however that counting the number of steps which satisfy some proposition is possible in our logic: it suffices to add one counter and increase it each time the proposition is satisfied. In this respect, we get some "parametric quantitative reasoning" ([14]) for free; the number of times some transition is fired can be a free variable in our logic.

We prove that recognizability by a flat automaton is equivalent to definability in $\mathcal{L}_p$. Note that this result goes both ways: unlike PLTL for which only star-free languages are definable, here, any flat language is definable (and conversely any definable language is flat). We prove that satisfiability of $\mathcal{L}_p$ formulas, as well as model-checking the negation of formulas in $\mathcal{L}_p$ (against a model described by a flat automaton) are decidable: this is a consequence of the relationship between flat formulas and flat automata on one hand and decidability results for flat automata on the other hand.

$\mathcal{L}_p$ has however several weaknesses. First, it is not closed by negation. This cannot be avoided as we show that validity of $\phi \in \mathcal{L}_p$ as well as model-checking ϕ are undecidable. Phrasing these results in term of automata, though emptiness is decidable for flat automata, the universality is undecidable.

$\mathcal{L}_p$ does not contain LTL. However, we can design a logic $\mathcal{L}_p^+$ which embeds both LTL and $\mathcal{L}_p$, while keeping the nice decidability properties. Now, instead of flat automata, each formula of $\mathcal{L}_p^+$ can be associated with a *piecewise flat automaton* which accepts the models of the formula. Emptiness remains decidable for such automata, which implies again that satisfiability and model-checking the negation of a formula are decidable (this includes reachability for instance).

We start in section 2 by definitions and examples of (flat) counter automata. In section 3 we establish (un)decidability results for flat automata. The flat logic $\mathcal{L}_p$ is introduced in section 4 where we also prove the correspondence with flat automata. Then we consider in section 5 the decision problems for this logic. Finally, in section 6 we consider the extension $\mathcal{L}_p^+$ which also embeds LTL.

2 Flat Counter Automata

Our constraints relate the current values (unprimed variables) and the next values (primed variables) of the counters, in a declarative way.

Definition 1 (Constraint). *An* atomic constraint *is one of the expressions:* $x\#y + c$, $x\#c$, $c\#x$ *where* $\# \in \{\leq, <\}$ *and* $c \in \mathbb{Z}$ *(resp.* $c \in \mathbb{Q}$*). A* constraint c *is either the constant* **true**, *the constant* **false** *or a conjunction of atomic constraints. The set of constraints with free variables* $x_1, \ldots, x_k, x'_1, \ldots, x'_k$ *is written* $\mathcal{C}(x_1, \ldots, x_k)$.

A constraint c in $\mathcal{C}(x_1, \ldots, x_k)$ defines a binary relation R_c on D^k where $D \in \{\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \ldots\}$: the relational symbols $\leq, <$ are interpreted as the usual ordering, as well as constant addition. $(v, v') \in R_c$ iff the valuation in which the ith component of v is assigned to x_i and the ith component of v' is assigned to x'_i satisfies c.

Definition 2 (Counter automaton: syntax). *An* automaton with k counters *A is a tuple $(\Sigma, Q, q_0, F, \delta)$ where Q is a finite set of* states*, Σ is a finite alphabet, $q_0 \in Q$ is the* initial state*, $F \subseteq Q$ is the set of* final states *and $\delta \subseteq Q \times \mathcal{C}(x_1, \ldots, x_k) \times \Sigma \times Q$ is a* transition relation*. We write sometimes $q \xrightarrow[\mathcal{A}]{c,a} q'$ instead of $(q, c, a, q') \in \delta$.*

A configuration *of the automaton is a pair (q, v) with $q \in Q$ and $v \in \mathbb{N}^k$ (resp. $v \in \mathbb{Q}_+^k$). The automaton may* move *from a configuration (q, v) to a configuration (q', v') iff there is a transition $(q, \phi, a, q') \in \delta$ such that $v, v' \models \phi$: the free variables $x_1, \ldots, x_k$ are interpreted by $v_1, \ldots, v_k$ and the free variables $x'_1, \ldots, x'_k$ are interpreted by $v'_1, \ldots, v'_k$. We write $(q, v) \xrightarrow[\mathcal{A}]{a} (q', v')$ when the automaton $\mathcal{A}$ may move from a configuration (q, v) to a configuration (q', v') while reading a. a may be dropped if it is not relevant.*

Definition 3 (Counter automaton: semantics). *Let w be finite (resp. infinite) word of length $|w|$: $w \in (\Sigma \times \mathbb{N}^k)^*$ (resp. $w \in (\Sigma \times \mathbb{N}^k)^\omega$). A* run *of $\mathcal{A}$ on w is a finite (resp. infinite) word $\rho \in Q^*$ of length $|w|$ (resp. $\rho \in Q^\omega$) such that $\rho(1) = q_0$ and, for every $1 \leq i \leq |w|-1$ (resp. $i \geq 1$), $(\rho(i), v_i) \xrightarrow[\mathcal{A}]{} (\rho(i+1), v_{i+1})$ if $w(i) = (a_i, v_i)$.*

A run ρ is successful *if its last letter belongs to F (resp. if it contains infinitely many elements of F). A word w is* accepted *by $\mathcal{A}$ if there is a successful run of $\mathcal{A}$ on w.*

We write $L(\mathcal{A})$ the set of finite words accepted by $\mathcal{A}$ and $L_\omega(\mathcal{A})$ the language of infinite words accepted by $\mathcal{A}$.

Example 1. On figure 1 we have depicted a controller for a pay phone. There are two counters: x is the number of quarters which have been inserted and y measures the total communication time. We use the classical abbreviations: $x{+}{+}$ stands for $x' = x+1$ and $x{-}{-}$ stands for $x' = x-1$. Also, by convention, when x' (resp. y') is not present in a transition, the constraint $x' = x$ (resp. $y' = y$) is assumed.

Such an automaton is expected to interact with its environment; messages are followed either by a question mark, when they are received by the controller, or by an exclamation mark, when they are sent by the controller. These aspects are however irrelevant here.

The initial state (which is also the only final state) is q_1. A possible sequence of consecutive moves of the automaton is:

$$q_1, \begin{pmatrix}0\\0\end{pmatrix} \xrightarrow{lift?} q_2, \begin{pmatrix}0\\0\end{pmatrix} \xrightarrow{quarter?} q_2, \begin{pmatrix}1\\0\end{pmatrix} \xrightarrow{dial?} q_3, \begin{pmatrix}1\\0\end{pmatrix}$$
$$\xrightarrow{quarter?} q_3, \begin{pmatrix}2\\0\end{pmatrix} \xrightarrow{quarter?} q_3, \begin{pmatrix}3\\0\end{pmatrix} \xrightarrow{connected?} q_4, \begin{pmatrix}3\\0\end{pmatrix} \ldots$$

Note that, by choice of the final state, it is not possible to insert quarters forever.

Definition 4. *A counter automaton over a single letter alphabet ($|\Sigma| = 1$) is* flat *if there is an ordering on the states such that there is a possible move from some (q, v) to some (q', v') only if $q \geq q'$. Moreover, there is at most one transition from a state to itself.*

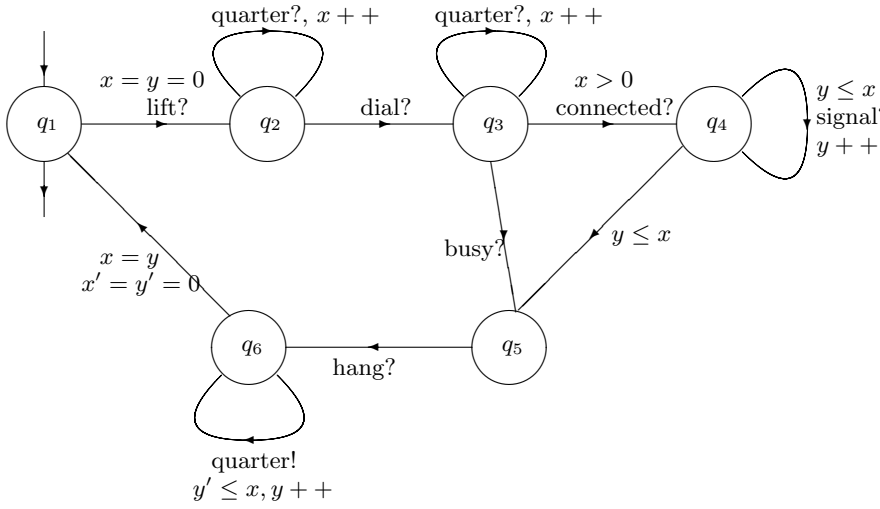


Fig. 1. A pay phone

Example 2. Consider the pay phone of figure 1 in which we forget the messages. The resulting automaton is not flat as there are several loops on a single state (e.g. q_2). It is however possible to replace each loop on a single state with a single transition, without changing the reachability relation. For instance the iteration of a loop labeled with $x++$ can be replaced with a single transition $x' > x$. Then the one step loops on q_2, q_3, q_4 and q_6 can be replaced with single transitions and the automaton becomes flat.

Also, if we remove the transition between q_6 and q_1, the automaton becomes flat.

3 (Un)decidability Results for Flat Counter Automata

We first recall here the decision results which can be derived from [9]. Then we prove new undecidability results.

Theorem 1 ([9]). *Given two states q_1, q_2 of a flat counter automaton A, there is an effectively computable formula of Presburger arithmetic $\phi_{q_1,q_2}(\boldsymbol{x}, n, \boldsymbol{x'})$ with $2k+1$ free variables such that $(q_1, v) \xrightarrow[A]{m} (q_2, v)$ iff $v, m, v' \models \phi_{q_1,q_2}$.*

where $\xrightarrow[A]{m} = \underbrace{\xrightarrow[A]{} \cdots \xrightarrow[A]{}}_{m}$.

Corollary 1 ([10]). *The emptiness of $L(A)$ (resp. $L_\omega(A)$) is decidable for flat automata A.*

Decidability of the emptiness of $L(A)$ follows directly from theorem 1: it suffices to decide $\exists m.q_0 \xrightarrow[A]{m} q_f$ for every final state q_f. Concerning $L_\omega(A)$, we need to decide the infinite iterability of a loop, which is also a consequence of the particular expression of the reachability relation, with some additional work [10].

Proposition 1. *The class of languages recognized by flat counter automata is effectively closed by union and intersection (both in the finite and in the infinite words cases).*

Proof sketch: The closure by union is straightforward. The closure by intersection is a consequence of the closure of $\mathcal{C}(x_1, \ldots, x_k)$ by conjunction. □

Unfortunately the class of languages recognized by flat automata is not closed under complement. Actually, we are going to show that the question of whether a flat automaton accepts all words in $(\mathbb{N}^k)^*$ is undecidable, which gives the non-closure results thanks to corollary 1.

First, consider the set CA_1 of counter automata over a one letter alphabet such that there is exactly one transition starting from a final state, which is labeled with **true**. The reachability of a final state in a Minsky machine reduces to the emptiness of the language recognized by such a counter automaton. Hence we have the undecidability result:

Lemma 1. *The emptiness problem for $L(A)$ (resp. $L_\omega(A)$) is undecidable for $A \in CA_1$.*

We may further restrict the class of counter machines, encoding the states into a counter. Let CA_2 be the class of automata in CA_1 which only contain two states q_1, q_2, such that q_2 is final and there is no transition from q_2 to q_1. (See figure 2.)

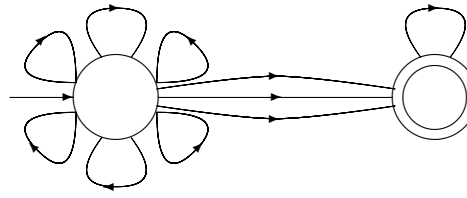


Fig. 2. An automaton in CA_2

If A is an automaton with $k+1$ counters and x is one particular counter then the *projection* $\pi_x(L(A))$ (resp. $\pi_x(L_\omega(A))$) is the subset of $(\mathbb{N}^k)^*$ (resp. $(N^k)^\omega$) of words in $L(A)$ in which the x component has been erased.

Lemma 2. *For every automaton $A \in CA_1$ with k counters $x_1, \ldots, x_k$, there is an automaton $A' \in CA_2$ with $k+1$ counters $c, x_1, \ldots, x_k$ such that $L(A) = \pi_c(L(A'))$ and there is a flat automaton A'' such that $L(A'') = (\mathbb{N}^k)^* - L(A')$ (resp. $L_\omega(A'') = (\mathbb{N}^k)^\omega - L_\omega(A')$)*

Proof sketch: First add a counter c which records the state number; without loss of generality, we may assume that numbering the states is such that $Q = \{q_1, \ldots, q_n\}$ and $Q_f = \{q_f, \ldots, q_n\}$ (i.e. states whose number is larger than f are final). The automaton A' contains two states: Q and Q_f. A transition from state i to state j with a constraint ϕ becomes, when i is not final (for instance), a constraint $\phi \wedge c = i \wedge c' = j$ from the initial state to itself, or to the final state if $q_j \in Q_f$.

Let $\phi_1, \ldots, \phi_n$ be the constraints of the transitions on the initial state and $\psi_1, \ldots, \psi_m$ be the constraints of the transitions from the initial to the final state in A'. Note that, by construction, for every i, $\psi_i \models c' \geq f$.

Let $g_1 \vee \ldots \vee g_r$ be a disjunction of constraints which is logically equivalent to

$$\neg((\bigvee_{i=1}^{n} \phi_i) \vee (\bigvee_{i=1}^{m} \psi_i))$$

Such a disjunction of constraints always exist since the negation of an atomic constraint can always be written as a disjunction of atomic constraints.

Our flat automaton is built as depicted on figure 3. A word which is not

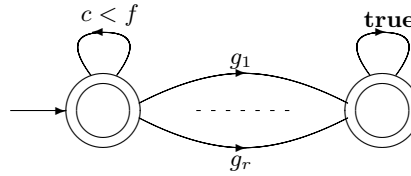


Fig. 3. The flat automaton in the proof of lemma 2

accepted either never reaches a final state, i.e. c remains strictly smaller than f, or else it is not compatible with the transition relation at some point, before reaching a final state. □

Lemma 2 is a little bit confusing; one may get the impression that the complement of any counter language (over a one letter alphabet) is a recognized by a flat automaton. This is not true, however; the projection plays an important role here. On the other hand, we know that the complement of a flat automaton, cannot be always recognized by a flat automaton: universality would then be decidable, hence the emptiness for any counter automaton.

From the two previous lemmas we can derive the following:

Theorem 2. *The universality is undecidable for flat automata (both in the case of finite and in the case of infinite words).*

4 The Flat Counter Logic $\mathcal{L}_p$

We introduce first a logic with counters CLTL, which, unfortunately, is too expressive. However, the notion of flat automaton which we introduced in the last section can be easily characterized at the logical level using a restriction of CLTL, which is similar to the so-called "flat fragment" in [12] for instance.

4.1 A Logic with Counters

Basically, we consider a temporal logic whose modalities are the same as in PLTL. The only difference is that, instead of propositional atomic formulas, we allow arbitrary constraints in $\mathcal{C}(x_1, \ldots, x_k)$.

More precisely, given a natural number k and a finite set of propositional variables $\mathcal{P}$, CLTL is the smallest set of formulas such that P belongs to CLTL for every $P \in \mathcal{P}$, $\mathcal{C}(x_1, \ldots, x_k)$ is included in CLTL and if ϕ_1 and ϕ_2 are formulas of CLTL, then $\phi_1 \wedge \phi_2, \phi_1 \vee \phi_2, \neg\phi_1, \mathbf{X}\phi_1, \phi_1 \,\mathcal{U}\, \phi_2$ are formulas of CLTL.

We may also use the classical derived operators $\Box$ ("henceforth") and $\Diamond$ ("eventually").

Temporal formulas are interpreted over *computations* which are now infinite words in $2^{\mathcal{P}} \times \mathbb{N}^k$. Given an infinite path $\pi \in (2^{\mathcal{P}} \times \mathbb{N}^k)^\omega$, we write $\pi(i)$ for the ith letter of π and we let $\widetilde{\pi}$ be the infinite word in $(2^{\mathcal{P}} \times \mathbb{N}^k \times \mathbb{N}^k)^\omega$ defined by:

$$(\pi(i) = (a, \boldsymbol{v}) \text{ and } \pi(i+1) = (b, \boldsymbol{w})) \text{ implies } \widetilde{\pi}(i) = (a, v, w)$$

This little technicality is necessary because the constraints may express relations between two successive values of the counters and not only constraints on a given value of the counters.

Now, a path π satisfies ϕ iff $\widetilde{\pi}, 0 \models \phi$ and:

- $\widetilde{\pi}, i \models \mathbf{true}$ and $\widetilde{\pi}, i \not\models \mathbf{false}$
- $\widetilde{\pi}, i \models P$ where $P \in \mathcal{P}$ if and only if $\pi(i) = (a, \boldsymbol{v})$ and $P \in a$
- $\widetilde{\pi}, i \models \phi(x_1, \ldots, x_k, x'_1, \ldots, x'_k)$ where $\phi \in \mathcal{C}(x_1, \ldots, x_k)$ iff $\widetilde{\pi}(i) = (a, \boldsymbol{v}, \boldsymbol{w})$ and $\boldsymbol{v}, \boldsymbol{w} \models \phi$ (with the usual definition of satisfaction in Presburger arithmetic).
- $\widetilde{\pi}, i \models \mathbf{X}\phi$ iff $\widetilde{\pi}, i+1 \models \phi$,
- $\widetilde{\pi}, i \models \phi_1 \wedge \phi_2$ iff $\widetilde{\pi}, i \models \phi_1$ and $\widetilde{\pi}, i \models \phi_2$, ...
- $\widetilde{\pi}, i \models \phi_1 \,\mathcal{U}\, \phi_2$ iff there is an index $j \geq i$ such that $\widetilde{\pi}, j \models \phi_2$ and for all $k \in [i, j[$, $\widetilde{\pi}, k \models \phi_1$.

Example 3. CLTL allows to express properties such as: "x is never greater than 100" or "each time x is larger than 100, an alarm is raised" or "ultimately, the register x remains stable" :

$$\Box(x \leq 100), \qquad \Box(x \leq 100) \vee (x \leq 100 \,\mathcal{U}\, alarm \geq 1) \qquad \Diamond\Box(x' = x)$$

Unfortunately, CLTL is too expressive:

Theorem 3. *Satisfiability is undecidable for CLTL. Model checking (of a flat automaton) is also undecidable in this logic.*

Proof sketch: We reduce the halting problem of a counter machine. Roughly, we use an auxiliary variable c ranging over the states of the machine and encode the computations of the machine by the formula:

$$c = q_0 \wedge \left(\left[\bigwedge_i (c = q_i \Rightarrow \bigvee_{q_i \xrightarrow{G} q_j} [G_{q_i,q_j}(\boldsymbol{x}, \boldsymbol{x'}) \wedge c' = q_j]) \right] \mathcal{U} \left[\bigvee_{q_f \in F} c = q_f \right] \right)$$

□

4.2 The Flat Fragment of the Logic

$\mathcal{L}_p$ is defined by a syntactic restriction of the formulas, which, roughly, restricts the left members of "until" to be conjunctions of atomic formulas, thus preventing the construction of theorem 3. For simplicity, we assume here that $\mathcal{P} = \emptyset$; propositional variables will be re-introduced in section 6 and, anyway, they can be encoded by integer variables.

Definition 5. *An* elementary formula *is a Boolean combination of constraints in* $\mathcal{C}(x_1, \ldots, x_k)$.

The set $\mathcal{L}_p$ *of flat formulas, is the smallest subset of CLTL such that:*

- *elementary formulas are flat*
- *if* ϕ_1, ϕ_2 *are flat, then* $\phi_1 \wedge \phi_2, \phi_1 \vee \phi_2$, $\mathbf{X}\phi_1$ *are flat.*
- *if* ϕ_1 *is a constraint in* $\mathcal{C}(x_1, \ldots, x_k)$ *and* ϕ_2 *is flat, then* $\phi_1 \,\mathcal{U} \phi_2$ *is flat*
- *if* ϕ *is a constraint in* $\mathcal{C}(x_1, \ldots, x_k)$*, then* $\neg(\mathbf{true}\,\mathcal{U} \neg\phi)$ *(i.e.* $\Box\phi$*) is flat*

The last condition is ad-hoc: it corresponds to the encoding of final states, as we will see.

Let us emphasize that $\mathcal{L}_p$ is *not* closed by negation. This is unavoidable as we will see in the next section. On the other hand, we could add the weak until, as both $\Box\phi$ and $\phi \,\mathcal{U} \psi$ are in $\mathcal{L}_p$ when ϕ is a constraint.

Example 4. The formulas given in example 3 are all flat.

One of the main interest of $\mathcal{L}_p$ is the correspondence with flat automata:

Theorem 4. *For every formula* ϕ *of* $\mathcal{L}_p$ *, there is a flat automaton which accepts the models of* ϕ*.*

Conversely, for every flat automaton $\mathcal{A}$*, there is a formula* ϕ *of* $\mathcal{L}_p$ *whose models are the words accepted by* $\mathcal{A}$*.*

Proof sketch: From logic to automata we use the closure properties of flat automata by union and intersection (theorem 1) and the standard constructions for $\mathcal{U}$, $\mathbf{X}$ and $\Box$. For instance consider $\phi\,\mathcal{U}\psi$. By hypothesis, ϕ belongs to $\mathcal{C}(x_1, \ldots, x_k)$. We construct the automaton for $\phi\,\mathcal{U}\psi$ by adding in front of the automaton for ψ a state on which there is a loop guarded by ϕ

From the automata to the logic, we proceed by induction on the ordering on states. From minimal states q there is at most one departing transition, say labeled with ϕ, and whose target is q itself. Then, if q is final, the corresponding formula will be $\Box\phi$ (**false** otherwise). For the induction step, if $q_1, \ldots, q_n$ are the successors of q and ϕ is the constraint of the loop on q, we get roughly the formula $\phi\,\mathcal{U}((\phi_1 \wedge \mathbf{X}\phi_{q_1}) \vee \ldots (\phi_n \wedge \mathbf{X}\phi_{q_n}))$. $\Box$

5 Satisfiability and Model-Checking in $\mathcal{L}_p$

Thanks to theorem 4 we can decide satisfiability and model checking of the negation of a formula of $\mathcal{L}_p$:

Theorem 5. *Given a formula $\phi \in \mathcal{L}_p$ and a flat automaton $\mathcal{A}$, the following questions are decidable:*

- *Is ϕ satisfiable ?*
- *Does $\mathcal{A}$ satisfy $\neg\phi$? (In other words, is there a word accepted by $\mathcal{A}$ which is a model of ϕ ?)*

Proof: Thanks to theorem 4, for every formula $\phi \in \mathcal{L}_p$, there is an automaton $\mathcal{A}_\phi$ which accepts the models of ϕ. Then satisfiability reduces to the emptiness of $L(\mathcal{A}_\phi)$ and $\mathcal{A} \models \neg\phi$ reduces to $L(\mathcal{A}) \cap L(\mathcal{A}_\phi) = \emptyset$. Now, thanks to theorems 1 and 1, both questions are decidable. $\Box$

Example 5. Negation of formulas in $\mathcal{L}_p$ include for instance reachability formulas $\Diamond q$ (adding here a new counter whose value is 0, except when reaching q) or safety formulas $\Box\neg\phi$ where ϕ is a constraint. Actually, considering the formulas in example 3, the negations of the first two formulas also belong to $\mathcal{L}_p$ because the negation of constraints $s \geq t$ are atomic constraints and the negation of $c\,\mathcal{U}c'$ is in $\mathcal{L}_p$ when c, c' are both of the form $s \geq t$. Only the negation of $\Diamond\Box x' = x$ is not a $\mathcal{L}_p$ formula.

It is also possible to reduce in polynomial time Presburger arithmetic satisfiability to $\mathcal{L}_p$ satisfiability, hence, in principle, $\mathcal{L}_p$ is at least as hard as Presburger arithmetic (between 2-DEXPTIME and 3-DEXPTIME).

Now, deciding $\mathcal{A} \models \phi$ for $\phi \in \mathcal{L}_p$ is equivalent to the decision of inclusion of flat automata, which is undecidable:

Theorem 6. *The validity problem and the model checking on a flat automaton are undecidable for a formula $\phi \in \mathcal{L}_p$.*

Sketch of the proof: This follows from theorems 4 and 2. $\Box$

6 $\mathcal{L}_p^+$: A Decidable Extension of $\mathcal{L}_p$ and *LTL*

The logic $\mathcal{L}_p$ is not fully satisfactory in many respects. In particular, the restrictions on the left member of an $\mathcal{U}$ disallow arbitrary LTL formulas. On the other hand, theorem 3 shows that we cannot simply drop the restriction. At least, we have to consider positive Boolean combinations of PLTL formulas and $\mathcal{L}_p$ formulas. We can still go a little further, as we will see.

Informally, $\mathcal{L}_p^+$ extends $\mathcal{L}_p$ by allowing any conjunction of a PLTL formula and a constraint where only constraints were allowed.

Definition 6 (Syntax of $\mathcal{L}_p^+$). *We assume given a finite set of propositional variables $\mathcal{P}$ and a positive integer k.*

Given a constraint ϕ, $PLTL_\phi$ is the smallest set of temporal formulas containing $\phi \wedge P_1 \wedge \ldots \wedge P_n \wedge \neg Q_1 \wedge \ldots \wedge \neg Q_m$ for every propositional variables $P_1, \ldots, P_n, Q_1, \ldots, Q_m$ and which is closed by $\wedge, \vee, \mathcal{U}, \mathbf{X}, \Box$. A basic formula *is a formula $\psi \in PLTL_\phi$ for some $\phi \in \mathcal{C}(x_1, \ldots, x_k)$.*

$\mathcal{L}_p^+$ is the smallest set of formulas such that:

- *every basic formula is in $\mathcal{L}_p^+$,*
- *if ϕ_1, ϕ_2 are in $\mathcal{L}_p^+$, then $\phi_1 \wedge \phi_2$, $\phi_1 \vee \phi_2$, $\mathbf{X}\phi_1$ are in $\mathcal{L}_p^+$*
- *if ϕ_1 is a basic formula and $\phi_2 \in \mathcal{L}_p^+$, then $\phi_1 \,\mathcal{U} \phi_2 \in \mathcal{L}_p^+$*
- *if ϕ_1 is a basic formula, then $\Box\phi_1 \in \mathcal{L}_p^+$.*

Note that, in PLTL, negation can be pushed to the propositional variables level if we include $\Box$ in the syntax. That is why PLTL formulas are basic formulas in the above definition: it is sufficient to choose $\phi = \mathbf{true}$. Constraints are also basic formulas, hence $\mathcal{L}_p^+$ is an extension of both $\mathcal{L}_p$ and PLTL.

On the other hand $\mathcal{L}_p^+$ is a fragment of the logic CLTL which was defined in section 4.1, from which we borrow the the semantics.

Example 6. We may record the elapsed time in a LTL formula using an auxilliary counter; for instance:

$$x = 0 \wedge ((p \wedge (x' = x + 1))\,\mathcal{U}(Q \wedge x' = x + 1))\,\mathcal{U}(R \vee x > \alpha)$$

is an $\mathcal{L}_p^+$formula, x recording the elapsed time. We could consider e.g. a second phase in R in which the time spent for each action is larger (or smaller), or even record something different, as, e.g., distance or available resources... However, it is not allowed to replace one of the two occurrences of $x + 1$ with $x + 2$: on the left of an until the constraint has to be the same everywhere.

Here, we have to extend the notion of a flat automaton, corresponding to the extension of the syntax of formulas.

Definition 7. *A* piecewise flat automaton *is a counter automaton on an alphabet $\Sigma = 2^{\mathcal{P}}$ such that there is a partition $Q_1 \uplus \ldots \uplus Q_m$ of the set of states Q and an ordering on $\{Q_1, \ldots, Q_n\}$ such that:*

- *for every i, there is a constraint $\phi_i \in \mathcal{C}(x_1, \ldots, x_k)$*

- *for every transition* $q \xrightarrow{c,a} q'$ *of the automaton, if* $q \in Q$ *and* $q' \in Q'$*, then* $Q \geq Q'$
- *for every transition* $q \xrightarrow{c,a} q'$ *such that* $q, q' \in Q_i$*, there is a conjunction* ψ *of proprositional variables and negations of propositional variables such that* $c = \phi_i \wedge \psi$

Example 7. Consider the pay phone example of figure 1. With each event, we associate a propositional variable. Then the behavior between two lift events (i.e. a "session") is described by a piecewise flat automaton. Actually, more complex actions could be described within the same class of models, for instance using more coins types, calling services...

Proposition 2. *The class of languages accepted by piecewise flat automata is closed under union and intersection.*

Sketch of the proof: It is almost the same as the closure of flat languages. We use the closure of $\mathcal{C}(x_1, \ldots, x_n)$ by conjunction and, for intersection, a product construction which is similar to the Büchi automata intersection construction. □

Theorem 7. *The models of an* $\mathcal{L}_p^+$ *formula are recognized by a piecewise flat automaton.*

Sketch of the proof: As before, we proceed by induction on the formula. Thanks to proposition 2, we only have to show the construction for $\mathbf{X}$ and $\mathcal{U}$. The construction for $\mathcal{U}$ is actually complicated. An example is depicted on figure 4. Let $\mathcal{A}_{\phi_1}$ be the automaton accepting the models of ϕ_1, Q_1 its set of states, and $\mathcal{A}_{\phi_2}$ be the automaton accepting the models of ϕ_2 and Q_2 its set of states.

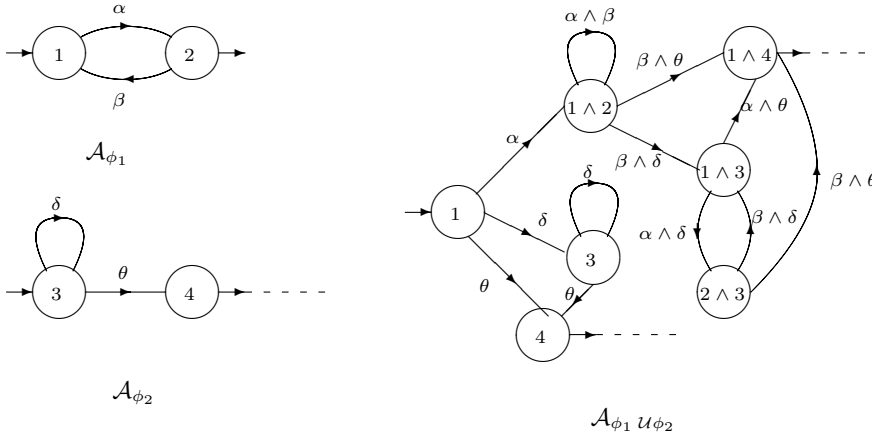


Fig. 4. The piecewise flat automaton for $\phi_1 \mathcal{U} \phi_2$

The idea is the following: while we do not reach a point where ϕ_2 is satisfied, at each move, the automaton launches a copy of $\mathcal{A}_{\phi_1}$ on the rest of the word. This is shown on an example on figure 4. Hence the set states of the automaton $\mathcal{A}_{\phi_1 \mathcal{U} \phi_2}$ is the union of $\{S \subseteq 2^{Q_1}, \mid q_0 \in S\}$ and $2^{Q_1} \times Q_2$, if q_0 is the initial state of $\mathcal{A}_{\phi_1}$.

The initial state of $\mathcal{A}_{\phi_1 \mathcal{U} \phi_2}$ is the singleton $\{q_0\}$, the final states are the pairs (S, q) where $S \subseteq Q_f^1$ and $q \in Q_f^2$, respectively the final states of $\mathcal{A}_{\phi_1}$ and $\mathcal{A}_{\phi_2}$. Transitions are computed as follows: a state $S \subseteq Q_1$ is considered as the conjunction for all states in S; if $S, S' \subseteq Q_1$, $q_0 \in S'$, f is a mapping from S to S', then there is a transition from S to S' which is labeled $\bigwedge_{q \in S} c_{q,f(q)}$ where $c_{q,f(q)}$ is the constraint of one of the transitions from q to $f(q)$ in $\mathcal{A}_{\phi_1}$. This corresponds to the case where we did not hit yet a position at which ϕ_2 is satisfied. We may also move from a state S to a state (S', q') if $q' \in Q_2$ and $S' \subseteq Q$, under the same conditions as above, except that we do not require $q_0 \in S'$ and move instead from the initial state of $\mathcal{A}_{\phi_2}$ to q' (see figure 4): this corresponds to the guess that we are going to satisfy ϕ_2 at the current position. Finally, we also have transitions from (S, q) to (S', q') which corresponds basically to the intersection of copies of $\mathcal{A}_{\phi_1}$ and one copy of $\mathcal{A}_{\phi_2}$.

The construction would be similar if we defined an alternating version of the automata and then transform it into a non-deterministic one: the exponential blow-up is unavoidable for the states of the formula ϕ_1.

One important remark is that we still get a piecewise flat automaton here, which would not be the case if we allowed arbitrary $\mathcal{L}_p^+$ formulas on the left of an until. Indeed, the powerset construction for $\mathcal{A}_{\phi_1}$ introduces transitions which are labeled with arbitrary conjunctions of constraints occurring in ϕ_1. It remains piecewise flat only because all these constraints are identical. □.

Theorem 8. *Emptiness is decidable for piecewise flat automata.*

Sketch of the proof: We only have to check the reachability for the projection automaton, where we forget the letters of Σ. Then all states in the same Q_i collapse into a single state and we are back to corollary 1. □

Theorem 9. *Satisfiability and model checking of $\neg\phi$ on A are decidable for $\phi \in \mathcal{L}_p^+$ an A a piecewise flat automaton.*

Sketch of the proof: This follows from theorems 8 and 7 and proposition 2. □

Finally, let us remark that we can also consider the conjunction of $\mathcal{L}_p^+$ formulas with arbitrary constraints in the additive theory of our domain D ($\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}_+$, $\mathbb{R}_+$). It is not difficult to see directly how satisfiability and model-checking can be decided, but there is one elegant way to do it:

Proposition 3. *For every formula ϕ in Presburger arithmetic, whose free variables are $x_1, \ldots, x_k$, there is a flat automaton $\mathcal{A}_\phi$ with $k+m$ counters such that,*

if E is the set of last letters of finite words accepted by $\mathcal{A}_\phi$, then

$$\{v \in \mathbb{N}^k, v \models \phi\} = \{v \in \mathbb{N}^k, \exists w \in \mathbb{N}^m, (v, w) \in E\}$$

Then we can build a piecewise flat counter automaton which accepts the models of both the $\mathcal{L}_p^+$ formula and the first-order constraint.

In other words, the proposition says that we can encode Presburger arithmetic in $\mathcal{L}_p^+$, which shows that we can perform some general parametric quantitative reasoning.

7 Conclusion

The symbolic representation of states played a crucial role in increasing the efficiency of model-checkers [8]. It is even more crucial for infinite states systems. We believe that *constraints*, i.e. logical formulas interpreted in a given domain, are an adequate symbolic representation in this case. The main advantage w.r.t. other representations is its declarativeness and the easy combination with logical formalisms.

In this paper, we provided with an example of application: we can design a temporal logic which combines the representation of infinite sets of configurations using constraints and the usual temporal properties. We have also shown a device (automaton) accepting the set of models, hence allowing to decide e.g. the satisfiability.

This generalizes the results on LTL satisfiability and model-checking: it is now possible to consider counters in a restricted way. Unlike in the previous works, we put the restrictions on the control of the automaton (flatness), which has a logical counterpart.

There is still one important weakness of our results: we do not know anything about their possible usefulness in practice. In principle, the complexity of the algorithms are prohibitive. However, the main source of complexity is the number of counters, which can be low (2 or 3) in many examples.

As we noticed at the end of the previous section, it is possible to express some parametric quantitative properties, as defined in [3,14] using additional counters and the logic $\mathcal{L}_p^+$. For instance, $\phi\ \mathcal{U}_{\leq x}\psi$ can be translated using an additional counter y into: $y = 0 \wedge ((\phi \wedge y' = y + 1)\ \mathcal{U}(y \leq x \wedge \psi))$. We want to investigate this application: which fragments of the PLTL logic of [3] are (easily) expressible in $\mathcal{L}_p^+$? For these fragments, we can check quantitative properties not only on finite automata, but also on piecewise flat automata with counters.

Another possible further investigation would be to consider the branching time temporal logic instead of PLTL.

References

1. P. A. Abdulla, A. Bouajjani, and B. Jonsson. On-the-fly analysis of systems with unbounded, lossy fifo channels. In *Proc. Computer Aided Verification*, volume 1427 of *Lecture Notes in Computer Science*, pages 305–318. Springer-Verlag, 1998.

2. R. Alur and D. Dill. Automata for modeling real-time systems. In *Proc. 17th Int. Coll. on Automata, Languages and Programming, Warwick, LNCS 443*, pages 322–335. Springer-Verlag, 1990.
3. R. Alur, K. Etessami, S. La Torre, and D. Peled. Parametric temporal logic for model measuring. In *Proc. Int. Conf. on Automata, Languages and Programming (ICALP'99)*, volume 1644 of *Lecture Notes in Computer Science*, pages 159–168, Prague, 1999. Springer-Verlag.
4. A. Bouajjani, R. Echahed, and P. Habermehl. On the verification problem of nonregular properties for nonregular processes. In *Tenth Annual IEEE Symposium on Logic in Computer Science*, pages 123–133, 1995.
5. A. Bouajjani, R. Echahed, and P. Habermehl. Verifying infinite state processes with sequential and parallel composition. In *Proc. POPL'95*, pages 95–106, San Francisco, 1995.
6. A. Bouajjani, R. Echahed, and R. Robbana. Verification of nonregular temporal properties of context free processes. In *Proc. CONCUR'94*, volume 836 of *Lecture Notes in Computer Science*, pages 81–97. Springer-Verlag, 1994.
7. A. Bouajjani and P. Habermehl. Symbolic reachability analysis of FIFO channel systems with non regular sets of configurations. In *Proc. 24th Int. Coll. on Automata, Languages and Programming (ICALP)*, volume 1256 of *Lecture Notes in Computer Science*, 1997.
8. J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, June 1992.
9. H. Comon and Y. Jurski. Multiple counters automata, safety analysis and presburger arithmetic. In A. Hu and M. Vardi, editors, *Proc. Computer Aided Verification*, volume 1427 of *LNCS*, pages 268–279, Vancouver, 1998. Springer-Verlag.
10. H. Comon and Y. Jurski. Counter automata, fixpoints and additive theories. Submitted to TCS. Available at http://www.lsv.ens-cachan.fr/∼comon/ftp.articles/mca.ps.gz, 1999.
11. H. Comon and Y. Jurski. Timed automata and the theory of real numbers. In *Proc. Conf. on Concurrency Theory (CONCUR)*, number 1664 in Lecture Notes in Computer Science, pages 242–257. Springer-Verlag, 1999.
12. D. R. Dams. Flat fragments of ctl and ctl*. *Journal of the IGPL*, 7(1):55–78, 1999.
13. E. Emerson and J. Y. Halpern. Sometimes and not never revisited. *J. ACM*, 33, 1986.
14. E. Emerson and R. Trefler. Parametric quantitative temporal reasoning. In *Proc. IEEE Symp. on Logic in Computer Science*, pages 336–343, Trento, 1999. IEEE Comp. Soc. Press.
15. J. Esparza. Decidability of model checking for infinite-state concurrent systems. *Acta Informatica*, 34:85–107, 1997.
16. Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems. Safety.* Springer-Verlag, 1995.
17. M. Minsky. *Computation, Finite and Infinite Machines.* Prentice Hall, 1967.
18. A. Sistla and E. M. Clarke. The complexity of propositional linear temporal logic. *J. ACM*, 32:733–749, 1985.
19. M. Vardi. An automata-theoretic approach to linear time temporal logic. In *Logic for concurrency: structure versus automata*, volume 1043 of *Lecture Notes in Computer Science*. Springer Verlag, 1996.

Sequents, Frames, and Completeness

Thierry Coquand[1] and Guo-Qiang Zhang[2,⋆]

[1] Department of Computer Science, University of Göteborg
S 412 96, Göteborg, Sweden
coquand@cs.chalmers.se
[2] Department of Computer Science, University of Georgia
Athens, GA 30602, U. S. A.
gqz@cs.uga.edu

Abstract. Entailment relations, originated from Scott, have been used for describing mathematical concepts constructively and for representing categories of domains. This paper gives an analysis of the freely generated frames from entailment relations. This way, we obtain completeness results under the unifying principle of the spatiality of coherence logic. In particular, the domain of disjunctive states, derived from the hyperresolution rule as used in disjunctive logic programs, can be seen as the frame freely generated from the *opposite* of a sequent structure. At the categorical level, we present equivalences among the categories of sequent structures, distributive lattices, and spectral locales using appropriate morphisms.

Introduction

Entailment relations were introduced by Scott as an abstract description of Gentzen's sequent calculus [15,16,17]. It can be seen as a generalisation of the earlier consequence calculus of Hertz [9] to a *multi-conclusion* consequence relation. The notion of consequence relation, with only one conclusion, has been analysed by Tarski [20]. This consequence calculus has been used by Scott in order to give a concrete representation of domains, as in information systems [18]. It is thus natural to wonder if the more general notion of entailment relation, with multiple conclusions, can be used to represent larger categories of domains, such as those related to non-determinism. This is indeed the case, and it has been developed in [21,22] and [5], in an independent way from Scott's work on entailment relations (in [21], a set together with an entailment relation is called a *sequent structure*). Another related reference, also independent from Scott's work, is [8].

In this paper we analyse various completeness theorems for sequent structures by embedding them into frames. A goal of this study is to provide a unified way to present completeness results in logic, such as those for resolution and hyperresolution.

⋆ Corresponding author.

P. Clote and H. Schwichtenberg (Eds.): CSL 2000, LNCS 1862, pp. 277–291, 2000.

A number of recent developments serve as the motivation for the current paper. In [3,4], it is shown that entailment relations are naturally connected to several mathematical structures. They can be used to give elegant constructive version of some basic mathematical concepts (and theorems), such as continuous linear forms, space of valuations, etc. One key point here is that it is often possible to get direct explicit descriptions of entailment relations generated by some rules, avoiding syntactical induction and case analysis on derivations.

In order to understand appropriate domains for the semantics of disjunctive logic programs, [23] introduces *clausal logic* based on the so-called hyperresolution rule [12]. Completeness of hyperresolution provides the basis for this domain-theoretic semantics: it establishes the equivalence of the model-theoretic semantics and the proof-theoretic semantics. Here, a set of clauses closed under hyperresolution is called a disjunctive state; the collection of disjunctive states under inclusion forms a complete lattice, which, in the case of information systems, is isomorphic to the Smyth powerdomain [13,23].

A natural question is whether the cpo of disjunctive states can be seen as a universal construction for sequent structures. Related to this question is the canonical embedding of a sequent structure into a frame. For this purpose we use Johnstone's coverage method [7] to study frames generated from a sequent structure as well as from its opposite. Interestingly, the frame generated from the opposite is precisely the complete lattice of disjunctive states. Moreover, in each case the universal map gives a way to capture a point of the frame as an ideal element of the underlying sequent structure.

The completeness theorem of coherent logic states that any coherent (or spectral) frame is *spatial* [7]. It ensures that enough models exist to uniquely determine the partial order, where models correspond to completely prime filters. This means that when sequent structures are embedded into spectral frames, we have enough models to uniquely determine the entailment relation, and thus obtain certain completeness result "for free", such as the completeness of hyperresolution. In return, existing results [13,23] related to hyperresolution suggest several explicit constructions for the sequent-structure-generated frames: a semantical one, a proof-theoretic one, and a third one based on the notion of "choice inference".

A couple of results in this paper may be seen as "folklore"; their roots may be traced back eventually to Stone's representation theorem [19]. We feel however that our contribution lies in tying in the more discrete notion of sequent structures with the more complete notion of locales through the so-called coverage relation [7] in a concrete logical setting. This allows the importation of existing results in locales to sequent structures, shedding new light on the topic. It is, for instance, quite interesting that the hyperresolution rule appears naturally in solving the problem of embedding an entailment relation in a frame, and it may not be obvious a priori that the disjunctive states form a frame. We hope that this paper is a first step in exploring completeness of various logical systems by means of canonical embedding to locales.

1 Coverage and Spatiality of Spectral Frames

A *frame* is a poset with finite meets and arbitrary joins which satisfies the infinite distributive law

$$x \wedge \bigvee Y = \bigvee \{x \wedge y \mid y \in Y\}.$$

For frames F and G, a frame morphism is a function $f : F \to G$ that preserves finite meet and arbitrary joins. Frames are also called *locales*.

Johnstone ([7], page 57) provides a way to construct a frame from a meet-semi-lattice based on the notion of *coverage relation*.

Definition 1. *Let $(S, \wedge, \leq)$ be a meet-semi-lattice. A* coverage *on S is a relation $\succ \subseteq 2^S \times S$ satisfying*

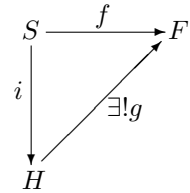
1. *if $Y \succ a$ then for any $y \in Y$, $y \leq a$.*
2. *if $Y \succ a$ then for any $b \leq a$, $\{y \wedge b \mid y \in Y\} \succ b$.*

A $\succ$-ideal determined by coverage $\succ$ is a subset I of S which is

1. *lower-closed: $a \in I$ & $b \leq a \Rightarrow b \in I$,*
2. *covered: $U \succ a$ & $U \subseteq I \Rightarrow a \in I$.*

A meet-semi-lattice S equipped with a coverage relation $\succ$ is called a *site*. A frame H with $i : S \to H$ is said to be *generated* from a site $(S, \succ)$ if

- i preserves finite meets,
- i transforms covers to joins: $Y \succ a \Rightarrow i(a) = \bigvee i(Y)$, and
- H, i is universal, i.e., the following diagram commutes:



Remark. For here and for the rest of the paper, all maps are assumed to preserve the respective structures they are acting on. This remark will be implicitly in force for all commutative diagrams and will not be repeated. We also remark in general that such a universal property guarantees that the generated structures are always *unique up to isomorphism*.

Here is Johnstone's basic result for the coverage relation.

Theorem 1 (Coverage Theorem [7], page 58). *The collection of $\succ$-ideals under inclusion is the frame generated from a site $(S, \succ)$.*

Recall that a frame can be seen as a "point-free" description of the open sets of a topological space. In this view, points are not basic, but are defined as collection of opens: a *point* of a frame is a completely prime filter, i.e. a filter α such that if $\bigvee X \in \alpha$ then there exists $x \in X$ such that $x \in \alpha$. If H is generated

from $(S, \succ)$ then a point is determined by its restriction to S, which is a filter α of S such that $a \in \alpha \ \& \ Y \succ a \ \Rightarrow \ (\exists b \in Y)\ b \in \alpha$. A frame H is called *spatial* (or has *enough points*) if for any $a, b \in H$, $a \leq b$ iff $\forall \alpha,\ a \in \alpha$ implies $b \in \alpha$, where α ranges over points of H. Intuitively, if we regard a, b as sets of points, then $a \leq b$ exactly when $a \subseteq b$.

There is a standard way to generate a frame from a distributive lattice D. One defines the coverage by letting $U \succ a$ if and only if $U \subseteq {\downarrow}a$ and there exists a finite subset X of U such that $a = \vee X$. By distributivity, this is a coverage relation. A $\succ$-ideal is then exactly an *ideal* of D: a downward-closed subset of D closed under finite joins. The generated frame is precisely the so-called *ideal completion* of D, which is written as $\mathsf{Idl}(D)$.

We say that a frame (locale) is *coherent* or *spectral* if it is isomorphic to the ideal completion of a distributive lattice[1]. The following fact will be used in the rest of the paper.

Theorem 2 (Page 65, [7]). *Spectral frames are spatial.*

2 Sequent Structures, Distributive Lattices, and Frames

We are interested in the question of frames generated by sequent structures. There are two ways to construct the frame generated by a sequent structure. The first construction, discussed in this section, is an implicit one built in two steps: obtaining the generated distributive lattice [3] first, and then taking its ideal completion as mentioned above. The second, explicit construction, is obtained by defining an appropriate coverage relation, which will be discussed in Section 5.

Let's recall the notion of entailment relation introduced by Scott in [15].

Definition 2. *An entailment relation (or a sequent structure) is a set A with a binary relation $\vdash$ between finite subsets* $\mathrm{Fin}(A)$ *of A such that*

$$(I) \qquad a \vdash a$$

$$(W) \qquad \frac{S \supseteq X \qquad X \vdash Y \qquad Y \subseteq T}{S \vdash T}$$

$$(C) \qquad \frac{X \vdash Y, a \qquad a, X \vdash Y}{X \vdash Y}$$

We use the notations $X, Y, \ldots$ for finite subsets of A, and X, Y for $X \cup Y$ while X, a for $X \cup \{a\}$.

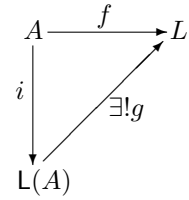
Several properties of entailment relations are self-evident. First, entailment relations are completely symmetric: $(A, \vdash)$ is an entailment relation iff $(A, \dashv)$ is. Second, entailment relations are closed under arbitrary intersections. Third, since the largest relation on $\mathrm{Fin}(A)$ is an entailment relation, given a family

[1] The term *coherent* is used in such a way in [7]. But it is used with another meaning in domain theory or even in [8]. The term *spectral*, used because such frames are exactly the ones that are spectrum of a commutative ring, is less ambiguous.

$(X_i, Y_i)_{i\in I}$ of pairs of finite subsets of A, the entailment relation *generated by the rules* $X_i \vdash Y_i$ can be seen to be the intersection of all entailment relations on A satisfying $X_i \vdash Y_i$ for all $i \in I$. (Of course one can close $(X_i, Y_i)_{i\in I}$ up by (I), (W), and (C) directly.) Last, information systems [18] can be seen as a special kind of entailment relation generated by rules of the form $X_i \vdash Y_i$ with Y_i being either a singleton or empty (intuitionistically they are more complex, however).

Distributive lattices freely generated from sequent structures make it possible to use lattice-theoretic constructions in sequent structures. The concept of freely generated lattices is introduced in [3].

Definition 3. *For a distributive lattice D and a sequent structure $(A, \vdash)$, a map $i : A \to D$ is said to preserve $\vdash$ if $X \vdash Y$ implies $\wedge i(X) \leq \vee i(Y)$. We say that the distributive lattice* $\mathsf{L}(A)$ *is* generated by $(A, \vdash)$ *if there is a $\vdash$-preserving map $i : A \to \mathsf{L}(A)$ which is universal among all such maps:*



Theorem 3 (Cederquist and Coquand [3]). *Any entailment relation $(A, \vdash)$ generates a distributive lattice $(\mathsf{L}(A), \leq)$ with a map $i : A \to \mathsf{L}(A)$ such that*

$$X \vdash Y \Leftrightarrow \wedge i(X) \leq \vee i(Y)$$

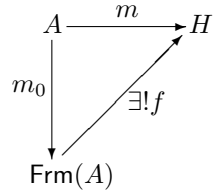
for all finite subsets X, Y of A, where $i(X)$ is the image of X under i.

We can study the similar topic of interpreting a sequent structure in a *frame*.

Definition 4. *Let H be a frame. An* interpretation *of a sequent structure $(A, \vdash)$ in H is a map $m : A \to H$ such that for every finite X, Y,*

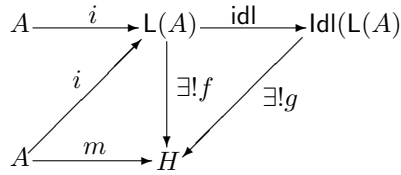
$$X \vdash Y \Rightarrow \wedge m(X) \leq \vee m(Y).$$

A frame $\mathsf{Frm}(A)$ *is* generated *by $(A, \vdash)$ if there is a universal interpretation $m_0 : A \to \mathsf{Frm}(A)$:*



Given a sequent structure $(A, \vdash)$, one can first generate the distributive lattice $\mathsf{L}(A)$ using Theorem 3 and then obtain the generated frame $\mathsf{Frm}(A) := \mathsf{Idl}(\mathsf{L}(A))$

by ideal completion (see the ending part of Section 1). Combining the two steps, we get the following commutative diagram:



This is exactly a proof of the following result:

Theorem 4. *Every sequent structure* $(A, \vdash)$ *generates a frame* $\mathsf{Idl}(\mathsf{L}(A))$ *with interpretation* $m_0 = \mathsf{idl} \circ i$.

One can prove additionally that the map m_0 has the property that for any finite X, Y, $X \vdash Y$ if and only if $\wedge m_0(X) \leq \vee m_0(Y)$. By Theorem 3 it is enough to show that if $\mathsf{idl}(u) \leq \mathsf{idl}(v)$ in $\mathsf{Idl}(\mathsf{L}(A))$ then $u \leq v$ in $\mathsf{L}(A)$, where $\mathsf{idl}(u)$ stands for the principal ideal generated by $u \in \mathsf{L}(A)$. But this follows from the special construction of $\mathsf{Idl}(\mathsf{L}(A))$ as the ideal completion of $\mathsf{L}(A)$.

3 Ideal Elements, Prime and Completely Prime Filters

What makes the results given in the previous section useful is that we have a canonical correspondence between ideal elements of the sequent structure, prime filters of the distributive lattice, and completely prime filters of the generated frame. We establish the correspondence in this section.

We must first recall what is an ideal element. Ideal elements have been used for representing domains. Given a sequent structure, the set of all of its ideal elements forms a dcpo under inclusion. One can obtain different categories of domains by considering different (sub)classes of sequent structures [22].

Definition 5. *A subset* $x \subseteq A$ *is called an ideal element with respect to a sequent structure* $\underline{A} = (A, \vdash)$ *if it is closed under entailment (where* $\subseteq^{\mathrm{fin}}$ *stands for "finite subset of"):*

$$(X \subseteq^{\mathrm{fin}} x \ \&\ X \vdash Y) \Rightarrow x \cap Y \text{ inhabited.}$$

The set of all ideal elements of $\underline{A}$ *is denoted as* $|\underline{A}|$.

A *co-element* of a sequent structure $(A, \vdash)$ is an ideal element of $(A, \dashv)$. By logical transposition, one easily checks classically that y is a co-element of $(A, \vdash)$ iff y is the complement of an ideal element x of $(A, \vdash)$; but our definition of co-element is formulated in a purely positive way.

As noted earlier, for any sequent structure $(A, \vdash)$, $(|\underline{A}|, \subseteq)$ is a dcpo (not necessarily with bottom).

Let $(A, \vdash)$ be a sequent structure and $A \xrightarrow{i} \mathsf{L}(A)$, $A \xrightarrow{m} \mathsf{Frm}(A)$ be the universal maps for the generated distributive lattice $\mathsf{L}(A)$ and generated frame $\mathsf{Frm}(A)$, respectively. For $x \subseteq A$, define $I_x \subseteq \mathsf{L}(A)$ as

$$I_x := \{u \in \mathsf{L}(A) \mid (\exists X \subseteq x) \ \wedge i(X) \leq u\}$$

and define $J_x \subseteq \mathsf{Frm}(A)$ in exactly the same way:

$$J_x := \{u \in \mathsf{Frm}(A) \mid (\exists X \subseteq x) \ \wedge m(X) \leq u\}.$$

We have the following result, which shows that ideal elements, prime filters, and completely prime filters uniquely determine each other under their respective universal maps. A direct proof of the second item is given later in Proposition 1.

Theorem 5. *Let $(A, \vdash)$ be a sequent structure.*

1. *If I is a prime filter of $\mathsf{L}(A)$ then the restriction of I to A, that is the set $i^{-1}(I)$, is an ideal element. Conversely if x is an ideal element of $(A, \vdash)$ then $I_x \subseteq \mathsf{L}(A)$ is a prime filter such that $x = i^{-1}(I_x)$.*
2. *If J is a completely prime filter of $\mathsf{Frm}(A)$ then the restriction of J to A, that is the set $m^{-1}(J)$, is an ideal element. Conversely if x is an ideal element of $(A, \vdash)$ then $J_x \subseteq \mathsf{Frm}(A)$ is a completely prime filter such that $x = m^{-1}(J_x)$.*

Proof. The first item is stated in [3] and the second item follows from item 1, Theorem 4, and an exercise in ([7], page 66) which states that there is a bijection between prime filters of $\mathsf{L}(A)$ and completely prime filters of $\mathsf{Frm}(A)$. ⊠

Note that ideal elements need not exist for an arbitrary sequent structure. In particular, if we allow $\emptyset \vdash \emptyset$, then there is no way to obtain an ideal element. However, we have this basic result:

Theorem 6 (Completeness). *Every sequent structure $(A, \vdash)$ has* enough *ideal elements: $X \vdash Y$* iff *for all ideal elements x, the set $x \cap Y$ is inhabited whenever $X \subseteq x$.*

This theorem is an immediate consequence of Theorem 2 and Theorem 5 above. A quite standard direct proof also exists by using classical logic and a weak form of the axiom of choice: one shows that if $X \nvdash Y$, then there is an ideal element x such that $X \subseteq x$ but $x \cap Y = \emptyset$. This is done by showing that the maximal filter F containing $\wedge X$ and disjoint from $\downarrow \vee Y$ in the generated lattice $\mathsf{L}(A)$ is prime. The ideas used in such a proof seem to come from Birkhoff [2].

It is worth noting a number of consequences of Theorem 6. First, if we start from a set of pairs $\{(X_i, Y_i) \mid i \in I\}$, then the least entailment relation generated by it can be described as $X \vdash Y$ if and only if for any x, if $X \subseteq x$, then $x \cap Y$ is inhabited, where x is an ideal element determined by $\{(X_i, Y_i) \mid i \in I\}$.

Secondly, as a special case of Theorem 6, we have $\emptyset \vdash \emptyset$ if and only if the sequent structure does not have any ideal element. This is precisely when the generated distributive lattice $\mathsf{L}(A)$ is *degenerated*, i.e., $0 = 1$. (However, a direct proof of this and the next remark is possible.)

Thirdly, from the proof of Theorem 6 we see that for any finite set $X \subseteq A$, there is an ideal element containing X if and only if $X \nvdash \emptyset$.

Finally, notice that rule (C) is a form of the resolution rule. Thus, we get as a consequence completeness of resolution: a clause $X \vdash Y$ is a *semantical* consequence of a set of rules $X_i \vdash Y_i$, that is is valid in any model satisfying these rules, iff it can be deduced from these rules using $(I), (W)$ and (C).

4 Clausal Logic and Hyperresolution

The notion of clause is a basic concept in logic programming. A natural framework for reasoning about clauses, called *clausal logic*, is demonstrated in [13,23] to play a fundamental role in disjunctive logic programming semantics.

With respect to a sequent structure $(A, \vdash)$, a *clause* is a finite subset of A, and a *clause set* is a collection of clauses. An ideal element x is a *model* of a clause u if $x \cap u \neq \emptyset$. x is a model of a clause set W if it is an model of every clause in W. There are three distinct notions of inference in clausal logic: $\models$, $\vdash^*_{hr}$, and the "choice inference" $\dashrightarrow$. For a clause set W and a clause u, we write

1. $W \models u$ if every model of W is a model of u. This is a model-theoretic concept, capturing the semantics.
2. $W \vdash^*_{hr} u$ if either $\emptyset \in W$, or u can be deduced from W using the so-called *hyperresolution rule*

$$\frac{a_1, X_1 \quad \ldots \quad a_n, X_n \qquad a_1, \ldots, a_n \vdash Y}{X_1, \ldots, X_n, Y}$$

   This is clearly a proof-theoretic, or operational, concept.
3. $\{X_1, \ldots, X_n\} \dashrightarrow u$ if $\{a_i \mid 1 \leq i \leq n\} \vdash u$ for any choice $a_1 \in X_1, a_2 \in X_2, \ldots, a_n \in X_n$. This is an intermediate notion: it uses the notion of arbitrary choice.

A result of [23] is that the three distinct notions of inference are equivalent to each other.

Theorem 7 (Rounds and Zhang). *Let $(A, \vdash)$ be a sequent structure. Let W be a finite clause set, and u a clause. The following three items are equivalent:*

1. $W \models u$,
2. $W \vdash^*_{hr} u$,
3. $W \dashrightarrow u$.

For any clause set C, we write $\Uparrow C$ for the least clause set containing C and closed under hyperresolution. A *disjunctive state* is a clause set C such that $C = \Uparrow C$.

The concept of disjunctive state is well-behaved on sequent structures [23]:

Theorem 8. *For a sequent structure $\underline{A}$, the set of all its disjunctive states under inclusion is a complete lattice.*

This theorem will be refined later, by giving a universal property of the lattice of disjunctive states w.r.t. the sequent structure $(A, \vdash)$.

5 Explicit Construction of Generated Frame Using Coverage

It is possible to give an explicit construction of the generated frame from a sequent structure $(A, \vdash)$ through an appropriate coverage relation defined by a dual form of hyperresolution.

For a sequent structure $(A, \vdash)$, consider the meet-semi-lattice $(\mathrm{Fin}(A), \cup, \supseteq)$ and the relation defined by $\{a_1, X,\ a_2, X, \ldots,\ a_n, X\} \succ X$ iff $X \vdash a_1, \ldots, a_n$. Note that no subscripts are used for the Xs here. Note also that if $X \vdash \emptyset$, then we have $\{\ \} \succ X$ (one can take this to be the $n = 0$ case). This is clearly a coverage relation, according to Definition 1. A $\succ$-ideal is, by definition, precisely a subset $U \subseteq \mathrm{Fin}(A)$ such that

- if $X \in U$ and $Y \supseteq X$, then $Y \in U$;
- if $\{a_1, X,\ a_2, X,\ \ldots,\ a_n, X\} \subseteq U$ and $X \vdash a_1, \ldots, a_n$, then $X \in U$.

We call such $\succ$-ideals *conjunctive states* and write H_0 for the set of all conjunctive states. For a set $U \subseteq \mathrm{Fin}(A)$, we write $\mathsf{c}U$ for the conjunctive state generated by U. Note that there is a conceptually simpler way to generate such a conjunctive state: first close U under finite super sets, and then add in all the Xs that are covered by some finite subset of the resulting set. We can do this because the only way to obtain a covered set is by removing at most one element from an existing set.

There is also a useful proof-theoretic reading of the generated conjunctive state. For any set $U \subseteq \mathrm{Fin}(A)$, its generated state $\mathsf{c}U$ consists of all Xs that can be derive from assumptions from U by using *supersets of sets* in U and the unique rule of inference:

$$\frac{a_1, X \quad \ldots \quad a_n, X}{X} \qquad \textit{provided} \quad X \vdash a_1, \ldots, a_n$$

By Theorem 1, we immediately obtain that the set of conjunctive states under inclusion is the frame generated from the meet-semi-lattice $(\mathrm{Fin}(A), \cup, \supseteq)$ with coverage $\succ$, which depends on $\vdash$. We show that this frame has the required universal property for an interpretation.

Lemma 1. *Let H be any frame. There is a bijection between (finite) meet-preserving maps $i : \mathrm{Fin}(A) \to H$ that transforms covers to joins, and interpretations $m : A \to H$.*

Proof. Suppose $i : \mathrm{Fin}(A) \to H$ preserves finite meets and transforms covers to joins. Define a map $m_i : A \to H$ by letting $m_i(a) := i(\{a\})$ for each $a \in A$. We show that m_i is an interpretation. Since i preserves finite meets and meet for $\mathrm{Fin}(A)$ is set union, we have, for any finite $X \subseteq A$,

$$i(X) = i(\bigcup_{a \in X} \{a\}) = \wedge_{a \in X} i(\{a\}) = \wedge_{a \in X} m_i(a) = \wedge m_i(X).$$

Suppose $X \vdash Y$, with $Y = \{a_1, \ldots, a_n\}$. By the definition of $\succ$,

$$\{a_1, X,\ a_2, X_2,\ \ldots,\ a_n, X\} \succ X.$$

Since i transforms covers to joins, we have

$$\begin{aligned} i(X) &= i(X \cup \{a_1\}) \vee \cdots \vee i(X \cup \{a_n\}) \\ &\leq i(\{a_1\}) \vee \cdots \vee i(\{a_n\}) \\ &= \vee m_i(Y). \end{aligned}$$

Therefore, $\wedge m_i(X) \leq \vee m_i(Y)$, as needed. (Note that when $X \vdash \emptyset$, the empty collection $\{\ \}$ covers X, by definition. Transforming covers to joins in this case means $i(X) = \bigvee \emptyset = 0$, which can be restated as $\wedge m_i(X) \leq \vee m_i(\emptyset)$.)

Suppose, on the other hand, that $m : A \to H$ is an interpretation. We define a map $i_m : \mathrm{Fin}(A) \to H$ by letting $i_m(X) := \wedge m(X)$ for each $X \in \mathrm{Fin}(A)$. By this definition, i_m automatically preserves finite meets. We show that it also transforms covers to joins. If $\{a_1, X,\ a_2, X,\ \ldots,\ a_n, X\} \succ X$ then by definition $X \vdash a_1, \ldots, a_n$. Therefore,

$$\wedge m(X) \leq m(\{a_1\}) \vee m(\{a_2\}) \vee \cdots \vee m(\{a_n\}).$$

By distributivity, we have

$$\wedge m(X) = (\wedge m(X \cup \{a_1\})) \vee \cdots \vee (\wedge m(X \cup \{a_n\})).$$

This means $i_m(X) = i_m(X \cup \{a_1\}) \vee \cdots \vee i_m(X \cup \{a_n\})$, which is exactly the required property of "transforming covers to joins".

It is clear that the given transformations $i \longmapsto m_i$ and $m \longmapsto i_m$ amount to a bijection. ⊠

By the previous lemma and the Coverage Theorem, we arrive at the next conclusion, which says that H_0 is the generated frame from $(A, \vdash)$.

Theorem 9. *For any sequent structure $(A, \vdash)$, the set of its conjunctive states H_0 is a frame under inclusion. Moreover, the interpretation $m_0 : A \to H_0$ mapping a to $\mathsf{c}\{a\}$ is universal. Furthermore we have $X \vdash Y$ if and only if $\wedge m_0(X) \leq \vee m_0(Y)$ for all finite subsets X, Y of A.*

Lemma 2. *Let $X, Y \in \mathrm{Fin}(A)$. Then $\mathsf{c}\{X\} \wedge \mathsf{c}\{Y\} = \mathsf{c}\{X\} \cap \mathsf{c}\{Y\} = \mathsf{c}\{X \cup Y\}$.*

Proof. We show the non-trivial part that $\mathsf{c}\{X\} \cap \mathsf{c}\{Y\} \subseteq \mathsf{c}\{X \cup Y\}$. Suppose $Z \in \mathsf{c}\{X\} \cap \mathsf{c}\{Y\}$. Then one has a derivation tree for Z with supersets of X as leaves/premises as well as a derivation tree for Z with supersets of Y as leaves. One can use structural induction on derivations to show that the two derivation trees can always be put together to obtain a derivation tree for Z with supersets of $X \cup Y$ as leaves. Therefore, $Z \in \mathsf{c}\{X \cup Y\}$. ⊠

The concrete notion of coverage $\succ$ allows a direct proof of the correspondence between ideal elements of $(A, \vdash)$ and completely prime filters of H_0, repeated as follows.

Proposition 1. *Let $m_0 : A \to H_0$ be the universal interpretation given in the previous theorem. If J is a completely prime filter of H_0 then the restriction of J to A, that is the set $m_0^{-1}(J)$, is an ideal element of $(A, \vdash)$. Conversely, if x is an ideal element then $J_x \subseteq H$ is a completely prime filter such that $x = m_0^{-1}(J_x)$, where $J_x := \{u \in H_0 \mid (\exists X \subseteq x) \ \wedge m_0(X) \leq u\}$.*

Proof. Suppose J is a complete prime filter of H_0. We show that $m_0^{-1}(J)$ is an ideal element. Suppose $X \vdash Y$ and $X \subseteq m_0^{-1}(J)$. Then $m_0(X) \subseteq J$ and so $\wedge m_0(X) \in J$ since J is a filter. Now Theorem 9 implies $\wedge m_0(X) \leq \vee m_0(Y)$, and so $\vee m_0(Y) \in J$. As J is prime, we have $m_0(b) \in J$ for some $b \in Y$. So $Y \cap m_0^{-1}(J)$ is inhabited.

On the other hand, suppose x is a ideal element. We show that J_x is a completely prime filter. It is easy to see that it is a filter. To show it is completely prime, we use the concrete representation of elements in H_0 as conjunctive states. By Lemma 2, $\wedge m_0(X) = \mathsf{c}\{X\}$. It suffices to show that if $X \subseteq x$ and $\mathsf{c}\{X\} \subseteq \mathsf{c}(\bigcup_{i \in I} u_i)$, where u_i are conjunctive states, then there exists some $Y \subseteq x$ and $i \in I$ such that $Y \in u_i$. For this it is enough to notice that whenever we can apply the rule

$$\frac{a_1, X \quad \ldots \quad a_n, X}{X} \qquad \textit{provided} \quad X \vdash a_1, \ldots, a_n$$

and $X \subseteq x$ then there exists i such that $X, a_i \subseteq x$. Indeed there exists i such that $a_i \in x$ because x is an ideal element. ⊠

As a result of Theorem 9, we can talk about joins and meets of finite subsets of A, with the understanding that such operations are always carried out in the generated frame, H_0 (or in the generated lattice $\mathsf{L}(A)$). *This is indeed the notational convention we adopt for the rest of the paper*: $\wedge X$ stands for $\wedge m_0(X)$.

Call $\wedge X$ a *semantical consequence* of $\wedge X_1, \ldots, \wedge X_n$ if for any ideal element x, $X \subseteq x$ implies $X_i \subseteq x$ for some i. We have the following completeness result, which is dual to Theorem 7.

Proposition 2. *Let H_0 be the frame generated by $(A, \vdash)$. The following are equivalent in H_0:*

1. *$\wedge X$ is a semantical consequence of $\wedge X_1, \ldots, \wedge X_n$.*
2. *$\wedge X \leq \wedge X_1 \vee \cdots \vee \wedge X_n$.*
3. *$X \vdash a_1, \ldots, a_n$ for any choice $a_1 \in X_1, \ldots, a_n \in X_n$.*

If we apply the construction H_0 to the opposite of the relation $\vdash$, which is also an entailment relation, but still use the same underlying meet-semi-lattice, we get the following result.

Theorem 10. *The complete lattice of all disjunctive states of $(A, \vdash)$ is the frame generated by $(A, \dashv)$.*

Proof. The elements of the frame generated by $\dashv$ are sets U of finite sets of A such that $X \in U$ whenever we have $X_1, \ldots, X_n \in U$ with $\vee X_1 \wedge \cdots \wedge \vee X_n \leq \vee X$. This is the same as the complete lattice of disjunctive states (see Section 4). ⊠

In particular, there is a canonical correspondence between *points* of the frame of all disjunctive states and *co-elements* of the sequent structure $(A, \vdash)$.

It is clear that the hyperresolution rule (Section 4) is equivalent to the rule

$$\frac{a_1, X \quad \dots \quad a_n, X}{X} \qquad \textit{provided} \quad a_1, \dots, a_n \vdash X$$

together with the rule

$$\frac{X}{Y} \qquad \textit{provided} \quad X \subseteq Y.$$

A simple combinatorial argument on permutation of rules show that we can even suppose the use of this last rule limited to the leaves of the derivation tree.

By duality, it follows from our results that X is derived by hyperresolution from $X_1, \dots, X_n$ iff

$$\vee X_1 \wedge \dots \wedge \vee X_n \leq \vee X$$

holds in D or equivalently, in H_0. Using Theorem 2 for the spectral frame H_0, this is true if and only if any point of H_0 containing $\vee X_1, \dots, \vee X_n$ contains also $\vee X$, which means exactly that the clause X is a semantical consequence (Section 4) of the clauses $X_1, \dots, X_n$. We get in this way yet another derivation of the completeness of the hyperresolution rule, Theorem 7 (see [12,23] as well). By soundness of the cut rule (C), which is nothing else than a form of the resolution rule, this gives a constructive proof that transforms any resolution proof into a hyperresolution proof.

In particular this shows the equivalence between $\vdash^*_{hr}$ and the "choice inference" $\dashrightarrow$, as stated in Theorem 7. There is, however, a direct proof of this equivalence.

Proposition 3. *We have $X_1, \dots, X_n \dashrightarrow X$ if and only if X follows from $X_1, \dots, X_n$ by the hyperresolution rule.*

Proof. For the "if" part we refer to [23]. We prove the "only if" part by induction on the size $\Sigma|X_i|$. Let $a_1 \in X_1, \dots, a_n \in X_n$. We claim that we can deduce all the clauses X, a_i $(1 \leq i \leq n)$ from $X_1, \dots, X_n$ using the hyperresolution rule. The result follows then from

$$\frac{a_1, X \quad \dots \quad a_n, X}{X} \qquad \textit{provided } a_1, \dots, a_n \vdash X$$

Let us prove X, a_1 from $X_1, \dots, X_n$; the other cases are similar. Notice that we have $b_1, \dots, b_n \vdash X, a_1$ for any choice $b_1 \in X - \{a_1\}, b_2 \in X_2, \dots, b_n \in X_n$. By induction hypothesis, we can deduce X, a_1 from $X_1 - \{a_1\}, X_2, \dots, X_n$ and hence from $X_1, \dots, X_n$. ⊠

6 Example: Spectrum of a Ring

Let us give an example in algebra, that illustrates some of the notions introduced here.

Let A be a commutative ring, and consider the entailment relation generated by the axioms

- $\vdash 0$,
- $x \vdash xy$
- $x, y \vdash x + y$
- $xy \vdash x, y$
- $1 \vdash$

We have the following direct description of $\vdash$.

Theorem 11. *$X \vdash Y$ if and only if the product of elements in Y belong to the radical of the ideal generated by X.*

Proof. We prove first that the relation "the product of elements in Y belong to the radical of the ideal generated by X" is an entailment relation, which satisfies all the rules above. We analyse only the rule (C), the other rules being directly checked: assume that we have both $X \vdash Y, a$ and $a, X \vdash Y$. Let y be the product of the elements in Y and I the ideal generated by X. We reason in A/I: by assumption ya is nilpotent (in A/I) and y belongs to the radical of the ideal generated by a. So we have m, n and x such that $y^n = ax$ and $(ya)^m = 0$. This implies $y^m(ax)^m = y^{mn+m} = 0$ and hence y is nilpotent in A/I. Hence $X \vdash Y$ as required.

It is direct that this entailment relation satisfies all the rules above.

Conversely, if the product of elements in Y belong to the radical of the ideal generated by X, we can derive $X \vdash Y$ using only the given axioms. Indeed, the first third rules show that $X \vdash y$ whenever y belongs to the ideal generated by X, while the two last rules show $y_1 \dots y_m \vdash y_1, \dots, y_m$. ⊠

In the particular case where A is a ring of polynomials, notice that we recover "for free" the proof of the formal Nullstellensatz theorem presented in [10]: the following items

- $x_1, \dots, x_n \vdash y$ is a consequence of the above axioms,
- y belongs to the radical of the ideal generated by $x_1, \dots, x_n$,
- $\{y\}$ can be derived from $\{x_1\}, \dots, \{x_n\}$ by hyperresolution

are equivalent.

An ideal element of this entailment relation is then exactly a proper prime ideal of A. Furthermore, if I is a radical ideal of A, then the set of finite subsets whose product is in I is a disjunctive state U_I. Conversely, if U is a disjunctive state, and I is the set of elements x such that $\{x\} \in U$ then I is a radical ideal such that $U = U_I$.

7 Categorical Equivalences

We extend our terminology first in order to adequately express categorical concepts related to sequent structures.

We have a natural category **Seq** of sequent structures, where a map $f : A \to B$ is simply a map which preserves entailment: $X_1 \vdash X_2$ in A implies $f(X_1) \vdash f(X_2)$ in B.

Furthermore, any distributive lattice D (and hence any frame) defines a sequent structure $G(D)$ by taking $X \vdash Y$ to mean $\wedge X \leq \vee Y$. This defines a functor $G : \mathbf{Spec} \to \mathbf{Seq}$ from the category of spectral frames and an interpretation $m : A \to H$ is nothing else than a map $A \to G(D)$ in the category **Seq**.

If A, B are sequent structures, we define an *approximable relation* from A to B to be an interpretation $m : A \to \mathsf{Frm}(B)$ of A in the frame generated by B. Notice that, in view of Theorem 9, this can be seen as a relation $\vdash$ between finite subsets of B and elements of A satisfying the following conditions:

- for any $x \in A$ the set of all $Y \subseteq B$ such that $Y \vdash x$ is a conjunctive state,
- if we have $x_1, \ldots, x_n \vdash_A u_1, \ldots, u_m$ and $Y \vdash x_i,\ 1 \leq i \leq n$ then there exists $Y_1, \ldots, Y_m$ such that $Y_j \vdash u_j,\ 1 \leq j \leq m$ and $Y \vdash_B y_1, \ldots, y_m$ for any choice $y_j \in Y_j,\ j = 1, \ldots, m$.

By standard categorical construction (see for instance [11], Chapter VI, 5) we get that sequent structures with approximable maps form a category **RelSeq**.

Similarly, we can introduce the category **RelLat** of distributive lattices, and maps $m : D \to \mathsf{Idl}(E)$, where $\mathsf{Idl}(E)$ is the frame generated by E.

Theorem 12. *The categories* $\mathbf{RelSeq}, \mathbf{RelLat}, \mathbf{Spec}$ *are equivalent.*

Proof. The equivalence between $\mathbf{RelLat}, \mathbf{Spec}$ is standard (see [1], page 120), while the equivalence between **RelSeq** and **Spec** follows from the universal properties of the free frame construction (see for instance [11], Chapter VI, 5, Exercise 2). ⊠

8 Concluding Remarks

Sequent structures are the skeletons of propositional theories. A propositional theory can be reduced to a sequent structure by translating an entailment instance $\varphi_1 \vee \varphi_2 \vdash \psi_1 \wedge \psi_2$ to simpler ones $\varphi_i \vdash \psi_j$ $(i, j \in \{1, 2\})$ repeatedly until only $\wedge$ appear on the left, and only $\vee$ appear on the right (distributivity is used in this process). The remaining $\wedge$'s and $\vee$'s can then be removed by virtue of sequents. Of course this process can be reversed; but we believe that working at the sequent level can in many cases avoid tedious syntactic details.

It is possible to provide a similar treatment to infinitary sequent structures. These structures consist rules of the form $X \vdash Y$, with X finite and Y arbitrary. Any such structure can still be canonically embedded into a frame. However, completeness and compactness fail in this case. Except for the purpose of representing L-domains [22] and of providing a connection to sober spaces, the significance of such a concept remains to be seen. We omit the treatment of them due to space limitations.

We end by repeating the hope given in the introduction that this paper be a first step in exploring completeness of various logical systems by means of canonical embedding to locales. It should be interesting to develop richer tools for this purposes, in order to handle additional logical operators. The well-known Henkin construction for instance, has been investigated in this setting [14] for linear logic.

Acknowledgment. We would like to thank the anonymous referees for insightful comments which lead to the improved presentation.

References

1. S. Abramsky and A. Jung, Domain theory, in: *Handbook of Logic in Computer Science*, Vol 3, (Clarendon Press, 1995).
2. G. Birkhoff. On the combination of subalgebras. Proc. Camb. Philos. Soc. 29, 441-464, 1933.
3. J. Cederquist and Th. Coquand. Entailment relations and distributive lattices. To appear in the Proceedings of Logic Colloquium 98.
4. Th. Coquand and H. Persson. Valuations and Dedekind Prague Theorem. To appear in the Journal of Pure and Applied Logic.
5. M. Droste and R. Göbel. Non-deterministic information systems and their domains. *Theoretical Computer Science* 75, 289-309, 1990.
6. M. Fourman, R. Grayson. Formal Spaces. in L. E. J. Brouwer Centenary Symposium (Noordwijkerhout, 1981), 107-122, North-Holland, Amsterdam-New York, 1982.
7. P. Johnstone. Stone Spaces. Cambridge University Press, 1982.
8. A. Jung, M.A.M. Moshier and M. Kegelmann. Multi lingual sequent calculus and coherent spaces. Fundamenta Informaticae, vol 37, 1999, pages 369-412.
9. G. Gentzen Collected Works. Edited by Szabo, Not-Holland, 1969.
10. V. Lifschitz. Semantical completeness theorems in logic and algebra. Proc. Am. Math. Soc., vol. 79, 1980, p. 89-96.
11. S. MacLane, Categories for the working mathematician. Springer-Verlag, 1971.
12. J.A. Robinson. The generalised resolution principle. Machine Intelligence, vol. 3, p. 77-93. 1968.
13. W. Rounds and G.-Q. Zhang. Clausal logic and logic programming in algebraic domains. Submitted. Copy at: `http://www.cs.uga.edu/~gqz`
14. G. Sambin. Pretopologies and completeness proofs. J. Symbolic Logic 60 (1995), no. 3, 861-878.
15. D. Scott. Completeness and axiomatizability. Proceedings of the Tarski Symposium, 1974, p. 411-435.
16. D. Scott. Background to formalisation. in Truth, Syntax and Modality, H. Leblanc, ed., p. 411-435, 1973.
17. D. Scott. On engendring an illusion of understanding. Journal of Philosophy, p. 787-807, 1971.
18. D. Scott. Domains for denotational semantics. in: *Lecture Notes in Computer Science* **140**, 577-613, 1982.
19. M. H. Stone. The theory of representations for Boolean algebras. *Trans. Amer. Math. Soc.* 40, 37-111, 1936.
20. A. Tarski. *Logic, semantics, metamathematics.* Oxford, 1956.
21. G.-Q. Zhang. *Logic of Domains*. Birkhauser Boston, Inc., Boston, MA, 1991.
22. G.-Q. Zhang. Disjunctive systems and L-domains. *19th International Colloquium on Automata, Languages, and Programming*(ICALP'92), Lecture Notes in Computer Science 623, 1992, pp. 284-295.
23. G.-Q. Zhang and W. C. Rounds. An information-system representation of the Smyth powerdomain. International Symposium on Domain Theory. Shanghai, China, October 1999. Copy at: `http://www.cs.uga.edu/~gqz`

Disjunctive Tautologies as Synchronisation Schemes

Vincent Danos and Jean-Louis Krivine

Université Paris 7
Équipe Preuves, Programmes, Systèmes
2 place Jussieu, 75251 Paris Cedex 05, France

Abstract. In the ambient logic of classical second order propositional calculus, we solve the specification problem for a family of excluded middle like tautologies. These are shown to be realized by sequential simulations of specific communication schemes for which they provide a safe typing mechanism.

1 Introduction

Since the inception of the proof/program correspondence with the Curry-Howard isomorphism, one of the goals of proof-theory has been the interpretation of logical rules as programming instructions. Only recently has this correspondence been extended to *classical logic*, which is now explained as a typing system for a λ-calculus augmented with a 'call-with-current-continuation' primitive, or some similar form of control such as an 'exception handler'. This extremely interesting explanation, first promoted by Griffin [3] and Felleisen, now admits many variants, as well as neat proof-theoretic renderings by Parigot [6] and Girard [2], for instance. All these variants are given in terms of *sequential languages*, though.

In this paper, we significantly depart from this tradition by interpreting a family of classical formulas, namely pure *disjunctions of literals*, as specifying *synchronisation protocols*. We contend, more generally, that this paradigmatic shift towards a concurrent reading of classical logic, gives rise to illuminating behavioral explanations.

The programming language we'll be using to make our point is a concurrent extension of a variant of Felleisen's λC-calculus, so that the world of programs that formulas will be referring to will indeed be a world of concurrent processes. The ambient logic, *i.e.*, the means of expressing behavioural specifications, will be classical second order propositional logic, equipped with only the following logical operators: $\rightarrow$ and $\forall$. More comprehensive frames, such as second order predicate calculus or even Zermelo-Frænkel set theory, are amenable to the same treatment, as shown in [4], but this simple logic suffices for our present purposes. Finally, the main and only tool we shall use is a revised version of the second

P. Clote and H. Schwichtenberg (Eds.): CSL 2000, LNCS 1862, pp. 292–301, 2000.

author's *realizability* method, which itself is an adaptation of the Tait-Girard reducibility method.

The gist of the interpretation is best understood with a simple example. Let's consider the formula $G_1 = \forall R \forall S\,[(R \to S) \vee (S \to R)]$. For one thing, we may recode implication as a disjunction and get a classically equivalent form of G_1, which is a pure disjunction of literals, namely $G_0 = \forall R \forall S\,[(\neg R \vee S) \vee (\neg S \vee R)]$. This last form is obviously true, so that G_1 itself is a tautology.

Conversely, the standard second order recoding of disjunction, yields a purely implicative formula, which is an intuitionistic equivalent of G_1, namely $G = \forall R \forall S \forall X\,[((R \to S) \to X) \to (((S \to R) \to X) \to X)]$. This last formula can as well be reformulated as a rule:

$$G\ \frac{\Gamma, R \to S \vdash X \qquad \Gamma, S \to R \vdash X}{\Gamma \vdash X}$$

The operational explanation developed in this paper for such rules is as follows. Suppose we have two processes $P_1[k_1]$, $P_2[k_2]$, both of type X, with free variables, or channels, k_1, k_2, of respective types $R \to S$ and $S \to R$. By the rule above we can build a compound process of type X, say $[P_1 \mid P_2]$, within which k_1, k_2 are bound. Close examination with the realizability tool shows that the computational behavior prescribed by G is the following: $[P_1 \mid P_2]$ starts running both P_1 and P_2 concurrently; if both get locked in requesting values for their free variables, k_1 and k_2, their states can then be described as $(k_1)v_R\pi_S$ and $(k_2)v_S\pi_R$, where v_R, v_S are some terms and π_R, π_S are some sequences, or stacks of terms; $[P_1 \mid P_2]$ then resumes the computation by running $(v_S)\pi_S$ and $(v_R)\pi_R$ concurrently.

That G is a tautology amounts to saying, according to our interpretation, that this cross communication scheme is well-typed, in that if, as expected, v_R, v_S are of respective types R and S, and if π_R, π_S are providing suitable environments in which to evaluate any term of respective types R and S, then both $(v_S)\pi_S$ and $(v_R)\pi_R$ will interact correctly.

This paper gives theoretical support to such concurrent computational explanations. A detailed construction of the framework, language, types and the realizability tool is given first. We then exercise that tool to extract concurrent behavioral specifications from a few tautologies, such as G, which is the simplest interesting example. Eventually, we home in on a quite general result explaining the family of *purely disjunctive* tautologies as synchronisation protocols.

2 Terms, Types and Models

In this preliminary section we first define our language, and the logic which types it, and then set up the suitable notion of realizability interpretation.

2.1 The Programming Language

We first define the set Λ of *terms*, denoted t, and the set Π of *stacks*, denoted π, with the following grammar:

$$t = x, (t)t, t \,|\, t, \lambda x.t, \kappa x.t, *_t, *_\pi$$
$$\pi = \epsilon, t \cdot \pi$$

We then define *executables* as finite multisets on $\Lambda \times \Pi$, still denoting $|$ the multiset constructor. By definition this constructor is commutative and associative. Those executables are equipped with an *evaluation* relation, written $\succ$, and defined as the smallest preorder on the set of executables which is compatible with $|$ and such that:

$$\begin{array}{ll}
(t)u, \pi \succ t, u \cdot \pi & (\text{PUSH}) \\
t \,|\, u, \pi \succ t, \pi \,|\, u, \pi & (\text{DIST}) \\
(\lambda x.t), u \cdot \pi \succ t[*_u/x], \pi & (\text{L-STORE}) \\
*_u, \pi \succ u, \pi & (\text{L-LOAD}) \\
(\kappa x.t), \pi \succ t[*_\pi/x], \pi & (\text{K-STORE}) \\
*_\pi, t \cdot \pi' \succ t, \pi & (\text{K-LOAD})
\end{array}$$

Note the analogy between the two binders, λx and κx. The first takes a snapshot, denoted $*_u$, of u, the current top element of the stack, stores it in x and pops the stack, while the second is taking a snapshot, denoted $*_\pi$, of the *whole* stack π, stores it in x as well, and leaves the stack intact. When $*_u$ comes in head position, it simply loads its value u, while $*_\pi$ throws the top element of the stack to its value.

The usual *cc* construction can be recovered as $\lambda h.\kappa k.(h)k$, one interest of our variant formulation being that the analogy just noted is made more obvious.

As an example, set $\delta = \kappa x.x$, then for any π, we get that loop:

$$(\delta)\delta, \pi \succ \delta, \delta \cdot \pi \succ *_{\delta \cdot \pi}, \delta \cdot \pi \succ \delta, \delta \cdot \pi.$$

2.2 The Typing System

Formulas or types, denoted A, B, ..., are here second order propositional formulas. Typing judgements of the form $x_1 : A_1, \ldots, x_n : A_n \vdash t : B$, where t is a term and A_1, ..., A_n, B are formulas, are generated by the following rules:

$$\text{ax}\ \frac{}{\Gamma, x : A \vdash x : A}\ \text{var}$$

$$\rightarrow\text{i}\ \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B}\ \text{abs} \qquad \rightarrow\text{e}\ \frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash (t)u : B}\ \text{app}$$

$$\forall\text{i}\ \frac{\Gamma \vdash t : A}{\Gamma \vdash t : \forall X A} \qquad \forall\text{e}\ \frac{\Gamma \vdash t : \forall X A}{\Gamma \vdash t : A[B/X]}$$

$$\text{peirce}\ \frac{\Gamma, x : A \to B \vdash t : A}{\Gamma \vdash \kappa x.t : A}\ \text{cc} \qquad \text{mix}\ \frac{\Gamma \vdash t : A \qquad \Gamma \vdash u : A}{\Gamma \vdash t \,|\, u : A}\ \text{par}$$

The quantification introduction rule, $\forall$i, is subject to the constraint that X is not free in the context Γ. The first five rules give a standard presentation, known as *natural deduction*, for second order propositional intuitionistic logic. Alongside with the sixth rule, known as *Peirce's law*, we get one possible natural deduction presentation of second order propositional classical logic. The last rule, or the *mix* rule, is just there to add expressive power on the terms side.

2.3 Truth Values and Models

Let $\perp\!\!\!\perp$ be a given set of executables, which, we assume throughout the paper, is closed by $\succ^{-1}$ and $|$. That is to say: 1) if $e \in \perp\!\!\!\perp$ and $e' \succ e$, then $e' \in \perp\!\!\!\perp$, and 2) if $e \in \perp\!\!\!\perp$ and $e' \in \perp\!\!\!\perp$, then $e \,|\, e' \in \perp\!\!\!\perp$.

For any set of stacks $\mathcal{Z}$, set $\mathcal{Z} \to \perp\!\!\!\perp$ to be the largest set of terms $\mathcal{X}$ such that $\mathcal{X} \times \mathcal{Z} \subset \perp\!\!\!\perp$. Any such set of terms, which can be written as $\mathcal{Z} \to \perp\!\!\!\perp$ for some set of stacks $\mathcal{Z}$, will be said to be a *truth value*. Two particular truth values are of special interest, the largest one $\Lambda = \emptyset \to \perp\!\!\!\perp$, and the smallest one, denoted $\bot = \Pi \to \perp\!\!\!\perp$. For any $t, \pi \in \perp\!\!\!\perp$, $(*_\pi)t \in \bot$, so $\bot$ is empty iff $\perp\!\!\!\perp$ is.

Given a choice of $\perp\!\!\!\perp$, we can extend any map $|.|_0^- : Var \to 2^\Pi$, from propositional variables to 2^Π, to a map $|.|^- : Form(2^\Pi) \to 2^\Pi$, from formulas with parameters in 2^Π to 2^Π, as follows:

$$\begin{aligned}
&|\mathcal{Z}|^- = \mathcal{Z}\\
&|X|^- = |X|_0^-\\
&|A \to B|^- = (|A|^- \to \perp\!\!\!\perp) \cdot |B|^-\\
&|\forall X A|^- = \cup_{\mathcal{Z}} |A[\mathcal{Z}/X]|^-
\end{aligned}$$

In the last clause, the union is meant to range over all subsets $\mathcal{Z}$ of Π. For instance, we get $|\forall X X|^- = \cup|\mathcal{Z}|^- = \cup\mathcal{Z} = \Pi$, so that $|\forall X X| = \bot$.

We then define the dual map $|.| : Form(2^\Pi) \to 2^\Lambda$ simply by putting:

$$|F| = |F|^- \to \perp\!\!\!\perp,$$

so $|F|$ is always a truth value. Moreover, when F is closed, its value, $|F|$, only depends on the choice of $\perp\!\!\!\perp$. Such valuations of classical formulas can be factorized through a 'not-not' translation to intuitionistic formulas.

When $\perp\!\!\!\perp = \emptyset$, it is easily seen that $|.|$ can only take two values, namely $\emptyset$ and Λ, and that, for any closed formula F, $|F| = \Lambda$ iff F is valid. In this special case the model collapses down to the usual notion of two-valued model. The generalization of this fact to any choice of $\perp\!\!\!\perp$ is known as the *adequacy* property:

Proposition 1 *Let $x_1 : A_1, \ldots, x_n : A_n \vdash t : B$ be derivable, $\perp\!\!\!\perp$ be any set of multisets on $\Lambda \times \Pi$ closed by $\succ^{-1}$ and $|$, and $|.|_0^-$ be any map from propositional*

variables to 2^{Π}*, then, for all* $v_1 \in |A_1|, \ldots, v_n \in |A_n|$*, and for all* $\pi \in |B|^{-}$*,* $t[v_1/x_1, \ldots, v_n/x_n], \pi \in \perp\!\!\!\perp$*.*

Proof. The proof is by induction on the typing derivation of t. To ease the reading of the proof, we'll simply write A and A^{-} for $|A|$ and $|A|^{-}$, and $t[v_i/x_i]$ for $t[v_1/x_1, \ldots, v_n/x_n]$. We also skip the axiom and quantifier rules, which are trivial, and use no assumption on $\perp\!\!\!\perp$.

1. Application: $\frac{\Gamma \vdash t : A \to B \quad \Gamma \vdash u : A}{\Gamma \vdash (t)u : B}$. Pick $v_i \in A_i$ and $\pi \in B^{-}$.

By induction $t[v_i/x_i] \in A \to B = A \cdot B^{-} \to \perp\!\!\!\perp$ and $u[v_i/x_i] \in A$, so that $t[v_i/x_i], u[v_i/x_i] \cdot \pi \in \perp\!\!\!\perp$. But:

$$(t)u[v_i/x_i], \pi = (t[v_i/x_i])u[v_i/x_i], \pi \succ t[v_i/x_i], u[v_i/x_i] \cdot \pi,$$

$\perp\!\!\!\perp$ being closed by (PUSH), we deduce $(t)u[v_i/x_i], \pi \in \perp\!\!\!\perp$, qed.

2. Abstraction: $\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B}$. Pick $v \in A$, $v_i \in A_i$, $\pi \in A^{-}$ and $\pi' \in B^{-}$.

We have $*_v, \pi \succ v, \pi$, $\perp\!\!\!\perp$ being closed by (L-LOAD), we deduce $*_v \in A$. So, by induction, $t[v_i/x_i][*_v/x], \pi' \in \perp\!\!\!\perp$. But:

$$(\lambda x.t)[v_i/x_i], v \cdot \pi' = (\lambda x.t[v_i/x_i]), v \cdot \pi' \succ t[v_i/x_i][*_v/x], \pi',$$

$\perp\!\!\!\perp$ being closed by (L-STORE), we deduce $(\lambda x.t)[v_i/x_i], v \cdot \pi' \in \perp\!\!\!\perp$, qed.

3. Peirce: $\frac{\Gamma, x : A \to B \vdash t : A}{\Gamma \vdash \kappa x.t : A}$. Pick $v \in A$, $v_i \in A_i$, $\pi \in A^{-}$ and $\pi' \in B^{-}$.

We have $*_\pi, v \cdot \pi' \succ v, \pi$, $\perp\!\!\!\perp$ being closed by (K-LOAD), we deduce $*_\pi \in A \to B$. So, by induction, $t[v_i/x_i][*_\pi/x], \pi \in \perp\!\!\!\perp$. But:

$$(\kappa x.t)[v_i/x_i], \pi = (\kappa x.t[v_i/x_i]), \pi \succ t[v_i/x_i][*_\pi/x], \pi,$$

$\perp\!\!\!\perp$ being closed by (K-STORE), we deduce $(\kappa x.t)[v_i/x_i], \pi \in \perp\!\!\!\perp$, qed.

4. Mix: $\frac{\Gamma \vdash t : A \quad \Gamma \vdash t' : A}{\Gamma \vdash t \mid t' : A}$.

Pick $v_i \in A_i$ and $\pi \in A^{-}$. By induction, $t[v_i/x_i], \pi$ and $t'[v_i/x_i], \pi \in \perp\!\!\!\perp$, so, $\perp\!\!\!\perp$ being closed by $\mid$, we get $t[v_i/x_i], \pi \mid t'[v_i/x_i], \pi \in \perp\!\!\!\perp$. But:

$$(t \mid t')[v_i/x_i], \pi = t[v_i/x_i] \mid t'[v_i/x_i], \pi \succ t[v_i/x_i], \pi \mid t'[v_i/x_i], \pi,$$

$\perp\!\!\!\perp$ being closed by (DIST), we deduce $(t \mid t')[v_i/x_i], \pi \in \perp\!\!\!\perp$. □

The proof is perfectly modular, each listed case calling on its own closure conditions on $\perp\!\!\!\perp$ which we have carefully recorded. Indeed, this proof, in some sense, describes a set of computation rules which are compatible with the typing system.

3 The Specification Problem

We can now put to good use this adequacy result. Careful choices of $\perp\!\!\!\perp$ result in proving normalizability results, which has been the traditional application of

adequation. Here, we'll be using it in a somewhat different way, to solve the so-called *specification* problem. That is, given an F, is there any computational behavior which all terms of type F have in common ?

3.1 Booleans

Let's start with a very simple example: $B = \forall X\,[X \to (X \to X)]$.

From now on, as in the adequation proof, we'll simply write A and A^- in place of $|A|$ and $|A|^-$. The particular choices we'll make for $\perp\!\!\!\perp$ will always be closures by $\succ^{-1}$ and $|$ of a given generating set of executables.

Proposition 2 *Let $\vdash t : B$ be derivable, then for all terms a, b and for all stack π, tab, π evaluates into a multiset on $\{(a, \pi), (b, \pi)\}$.*

Proof. Indeed, let a, b be terms and π be a stack. Take $\perp\!\!\!\perp$ to be the closure of $\{(a, \pi), (b, \pi)\}$ and set $X^- = \{\pi\}$. Then, a and $b \in X$, hence, by adequation, $t, a \cdot b \cdot \pi \in \perp\!\!\!\perp$, so that by (PUSH), $tab, \pi \in \perp\!\!\!\perp$. □

For instance $t = \lambda x.\lambda y.x \,|\, \lambda x.\lambda y.y : B$ does behave in this way, since $tab, \pi \succ a, \pi \,|\, b, \pi$.

One can refine a specification. Though we didn't develop the formal material pertaining to predicate calculus, the following should be pretty self-explanatory.

Consider a language $\mathcal{L}$ with two individual constants, 0 and 1, set:

$$Bx = \forall X\,[X0 \to (X1 \to Xx)]\,,$$

and suppose $\vdash t : B0$. Take then $\perp\!\!\!\perp$ to be the closure of $\{(a, \pi)\}$ and set $X0^- = \{\pi\}$ and $X1 = \Lambda$. Clearly $a \in X0$ and $b \in X1$, so that $t, a \cdot b \cdot \pi \in \perp\!\!\!\perp$, and hence, by (PUSH), $tab, \pi \in \perp\!\!\!\perp$. So we get:

Proposition 3 *Let $\vdash t : B0$ be derivable, then for all terms a, b and for all stack π: tab, π evaluates into a multiset on $\{(a, \pi)\}$.*

Informally, this last proposition says that any $t : B0$ behaves as a certain number of $\lambda x.\lambda y.x$ running concurrently. A property which one might think of as computational consistency for our system.

To recap, adequation gives a means of decoding the behavior specified by a given formula with respect to a given language. This here is the main thrust. Conversely, one can use it to refute typability. For instance, the proposition above shows in particular that $t = \lambda x.\lambda y.x \,|\, \lambda x.\lambda y.y$ can't be of type $B0$, nor can be any term with the same behaviour.

3.2 The Excluded Middle

Set $T = \forall X \forall R \forall S\,[((R \to S) \to X) \to ((R \to X) \to X)]$, which is the usual implicative coding of the *excluded middle*, $T_1 = \forall R \forall S\,[(R \to S) \vee R]$.

We want to show:

Proposition 4 *Let $\vdash c : T$ be derivable, then for all terms ρ, σ, r and for all stacks π, π_R and π_S, such that for all terms a and b, $\rho a, \pi \succ a, r \cdot \pi_S$ and $\sigma b, \pi \succ b, \pi_R$, $c\rho\sigma, \pi$ evaluates into a multiset on $\{(r, \pi_R)\}$.*

Proof. Let ρ, σ, r, π, π_R and π_S be as above. Take for $\perp\!\!\!\perp$ the closure of $\{(r, \pi_R)\}$, and set $R^- = \{\pi_R\}$, $S^- = \{\pi_S\}$ and $X^- = \{\pi\}$.

First, we have $r \in R$. Let now $a \in R \to S$, by definition $a, r \cdot \pi_S \in \perp\!\!\!\perp$, so by $\perp\!\!\!\perp$ being closed: $\rho, a \cdot \pi \in \perp\!\!\!\perp$, hence $\rho \in (R \to S) \to X$. Likewise, if $b \in R$, by definition, $b, \pi_R \in \perp\!\!\!\perp$, so by $\perp\!\!\!\perp$ being closed: $\sigma, b \cdot \pi \in \perp\!\!\!\perp$, hence $\sigma \in R \to X$. By adequation we get: $c, \rho \cdot \sigma \cdot \pi \in \perp\!\!\!\perp$. □

This specification can be rendered informally as follows: c launches two executables, or two independent processes, ρ and σ, passing over to each a variable, a and b respectively, and a same context π; if both processes stop on these variables, c sends ρ's top stack element to σ, via b, so that σ runs again, while ρ dies.

Let's run an example. We set $((c)\rho)\sigma = \kappa k.(\sigma)\kappa h.(k)(\rho)h$, and by typing $k : X \to R$ and $h : R \to S$, we do have $c\rho\sigma : X$ as it should. If ρ and σ behave as in the proposition, we then get the following interaction:

$$\begin{aligned} c\rho\sigma, \pi &\succ (\sigma)\kappa h.(*_\pi)(\rho)h, \pi \\ &\succ \kappa h.(*_\pi)(\rho)h, \pi_R \\ &\succ (*_\pi)(\rho)*_{\pi_R}, \pi_R \\ &\succ (\rho)*_{\pi_R}, \pi \\ &\succ *_{\pi_R}, r \cdot \pi_S \\ &\succ r, \pi_R, \end{aligned}$$

with the expected result. And thus we get one possible sequential implementation of the specification. It can be observed that, in this particular example, σ is run first, which means that T can't be understood as specifying an exception handling mechanism where σ would be the handler !

A particular case is when $\sigma = \lambda x.x$, which at the level of types means $X = R$. The behavior is now, if ever $\rho a, \pi \succ a, r \cdot \pi_S$, then $((c)\rho)\lambda x.x, \pi$ evaluates to a multiset on $\{(r, \pi_R)\}$. If we further assume that all terms involved are sequential, *i.e.*, none involve the | operator, then we simply get $((c)\rho)\lambda x.x, \pi \succ r, \pi$, which is the behaviour of *cc*. In that $T[R/X]$ is trivially equivalent to Peirce's law; this shouldn't be too much a surprise.

3.3 The Symmetric Excluded Middle

We return to G, already presented in the introduction. This formula is not intuitionistically valid, and the logic obtained when adding it to intuitionistic logic is that of formulas which are true in any linear Kripke model. What we set up to prove is:

Proposition 5 *Let $\vdash c : G$ be derivable, then for all terms ρ, σ, r et s and for all stacks π, π_R and π_S, such that for all terms a and b, $\rho a, \pi \succ a, r \cdot \pi_S$ and $\sigma b, \pi \succ b, s \cdot \pi_R$, $c\rho\sigma, \pi$ evaluates into a multiset on $\{(s, \pi_S), (r, \pi_R)\}$.*

Proof. Let ρ, σ, r, s, π, π_R and π_S be as above. Take for $\perp\!\!\!\perp$ the closure of $\{(s, \pi_S), (r, \pi_R)\}$, and set $R^- = \{\pi_R\}$, $S^- = \{\pi_S\}$ and $X^- = \{\pi\}$.

For one thing $r \in R$ and $s \in S$.

Let now $a \in R \to S$, by definition $a, r \cdot \pi_S \in \perp\!\!\!\perp$, so by $\perp\!\!\!\perp$ being closed: $\rho, a \cdot \pi \in \perp\!\!\!\perp$, hence $\rho \in (R \to S) \to X$. Symmetrically, $\sigma \in (S \to R) \to X$. Whence by adequation we get: $c, \rho \cdot \sigma \cdot \pi \in \perp\!\!\!\perp$. □

An example is $((c_R)\rho)\sigma = \kappa k^{X \to S}.(\rho)\lambda x^R.(k)(\sigma)\lambda y^S.x : X$ or c_S the symmetric form exchanging σ and ρ. When ρ and σ behave as in the proposition, $((c_R)\rho)\sigma, \pi \succ r \cdot \pi_R$, whereas $((c_S)\rho)\sigma, \pi \succ s \cdot \pi_S$.

In pure propositional calculus, there is no means to discriminate between these two behaviours. As in the case of the boolean type, we can refine the specification in predicate calculus, by decorating G as:

$$\forall R \forall S \left[(\forall x Rx \to \forall x Sx) \vee \forall x (Sx \to Rx) \right].$$

Then all terms of that type will behave as c_R does.

Note also that, since T trivially implies G, there is a form of compatibility between the two specifications.

We don't rerun the operational explanation given in the introduction. Let us observe, yet, that the proposition is not saying that any c of type G is implementing the cross communication mechanism. In fact that can't be the case since sequential proofs, such as the one above, just can't express it. But some proofs will, such as $c_S \,|\, c_R$. Admittedly, the implementation is not very elegant. The reasonable thing to do, then, seems to extend the language with a new suitable primitive. One subtle point yet is that the naïve rule:

$$[a, r \cdot \pi_S \,|\, b, s \cdot \pi_R \,|\, \ldots] \succ [r, \pi_R \,|\, s, \pi_S \,|\, \ldots],$$

would be wrong in that r, s might respectively contain a and b, so that both executables r, π_R and s, π_S might know *a and b* and could subsequently deadlock by calling on the same channel.

One other desirable thing would be a general specification result about T- and G-like formulas, perhaps giving some insight on why they should specify a means

of synchronisation, and explaining as well what other schemes are typable. This is the object of the next and last subsection.

3.4 Disjunctive Tautologies

Let's consider a finite family of formulas A_i, $i \in I$, each of the form:

$$A_i = B_{i1} \to (\dots (B_{in_i} \to C_i) \dots),$$

where B_{ij}s and C_is are all propositional variables, and set A to be the universal closure of $\vee_I A_i$. Such an A will be called a *purely disjunctive* formula.

We define the truth set of A, denoted $tr(A)$, to be the set of triples i, j, k such that $B_{ij} = C_k$. Rewriting all implications in A as disjunctions yields a classically equivalent formula which is the closure of a disjunction of literals, namely $(\vee \neg B_{ij}) \vee C_k$, whence A is valid iff its truth set is not empty.

Conversely, by the standard second order encoding of disjunctions as implications, we can obtain an intuitionistic equivalent of A, which we still denote A in the proposition below.

Both T_1 and G_1, we already ran into, are purely disjunctive; a simpler example is $V = \forall A\,[A \to A]$, and a longer one is:

$$W = \forall ABC\,[(A \to (B \to C)) \vee (C \to A) \vee (C \to B)]\,.$$

The special thing about them is:

Proposition 6 *Let A be a purely disjunctive formula and let $\vdash c : A$ be derivable, then for all terms ρ_i, b_{ij} and for all stacks π_i, such that for all terms a, $\rho_i a, \pi \succ a, b_{i1} \cdots b_{in_i} \cdot \pi_i$, $c\rho_1 \dots \rho_n, \pi$ evaluates into a multiset on $\{(b_{ij}, \pi_k) : i, j, k \in tr(A)\}$.*

Proof. Note that if A is false, then, hopefully, $\vdash c : A$ is not derivable, and the statement then vacuously holds.

The proof goes the usual way. Take for $\perp\!\!\!\perp$ the closure of $\{(b_{ij}, \pi_k) : i, j, k \in tr(A)\}$, set $X^- = \{\pi\}$ and $C_i^- = \{\pi_k : C_i = C_k\}$.

First we observe that $b_{ij}, \pi_k \in \perp\!\!\!\perp$ whenever $B_{ij} = C_k$, hence $b_{ij} \in B_{ij}$ if there is some k such that $i, j, k \in tr(A)$. If not, we simply take $B_{ij} = \Lambda$. In all cases, we now have $b_{ij} \in B_{ij}$.

Let $a \in A_i$, then $a, b_{i1} \cdots b_{in_i} \cdot \pi_i \in \perp\!\!\!\perp$, and so does $\rho_i a, \pi$, hence $\rho_i \in A_i \to X$. Therefore, $c\rho_1 \dots \rho_n, \pi \in \perp\!\!\!\perp$. □

If we consider any protocol generated by V, then we see it is sequential, c merely feeding in ρ with the identity, should ρ ever stop on a. In fact, V is intuitionistically valid, and any intuitionistic proof of a purely disjunctive formula generates

a dummy protocol, by the ... disjunction property. That is, intuitionistic terms, even using |, can only fork and will never synchronise back their threads. In fact, | itself behaves like this and can be thought of as a degenerated synchronisation scheme associated to the disjunctive formula $\top \vee \top$, where $\top$ stands for the logical constant 'true'.

Next, if we turn to W, we see that associated protocols can be non linear and non deterministic either. Specifically, if our three processes are blocked as in $\rho_1 a_1, \pi \succ a_1, v_A \cdot v_B \cdot \pi_C$, $\rho_2 a_2, \pi \succ a_2, v_C \cdot \pi_A$ and $\rho_3 a_3, \pi \succ a_3, v'_C \cdot \pi_B$, then communication could be implemented by launching concurrently v_C, π_C and v'_C, π_C, or by launching one of them only, etc. There is room in such a specification for creating dynamic patterns of synchronisation.

4 Conclusion

So what ? We have shown disjunctive tautologies have as realizers $\lambda\kappa$-terms which are dying to be read as sequential implementations of more abstract concurrent programs, namely synchronisation schemes. We didn't, as the referrees would point out, develop those concurrent programs in an independent syntax such as the pi- or the join-calculus [7]. That remains to be done. We didn't either bring in any concrete concurrent example backing the expressiveness of these schemes. As all synchronisations here are by construction deadlock-free they shouldn't be expected to have imme nse expressive power anyway.

References

1. V. Danos, J.-B. Joinet, H. Schellinx. A New Deconstructive Logic: Linear Logic (1996). *Journal of Symbolic Logic.*
2. J.-Y. Girard. A New Constructive Logic: Classical Logic (1992). *Mathematical Structures in Computer Science.*
3. T. Griffin. A formulae-as-types notion of control (1990). *In Proceedings of POPL'90.*
4. J.-L. Krivine. Typed Lambda-Calculus and Classical ZF Set-Theory (2000). *Archive for Mathematical Logic.*
5. L. Ong, C. Stewart. A Curry-Howard Foundation for Functional Computation with Control (1997). *In Proceedings of POPL'97.*
6. M. Parigot. Strong Normalization for Second-Order Lambda-Mu Calculus (1993). *In Proceedings of LICS'93.*
7. Moscova Project. The Join-Calculus Language: `http://pauillac.inria.fr/join/.`

Axiomatizing the Least Fixed Point Operation and Binary Supremum

Zoltán Ésik*

University of Szeged
Department of Computer Science
P.O.B. 652, 6701 Szeged, Hungary
esik@inf.u-szeged.hu

Abstract. The equational properties of the least fixed point operation on (ω-)continuous functions on (ω-)complete partially ordered sets are captured by the axioms of iteration algebras, or iteration theories. We show that the equational laws of the binary supremum operation in conjunction with the least fixed point operation on (ω-)continuous functions on (ω-)complete semilattices have a finite axiomatization over the equations of iteration algebras. As a byproduct of this relative axiomatizability result, we obtain complete infinite equational, and finite implicational axiomatizations.

1 Introduction

Consider the language of μ-terms given by the syntax

$$T ::= x \mid \sigma(\overbrace{T, \ldots, T}^{n-\text{times}}) \mid T + T \mid 0 \mid \mu x.T$$

where x ranges over a countably infinite set of variables, and for each $n \geq 0$, σ ranges over a set Σ_n of n-ary function symbols. Such terms may be interpreted as (ω)-continuous functions on (ω-)complete semilattices, or as monotonic functions on complete semilattices A, where $+$ denotes the supremum operation, 0 denotes the least element of A, and where terms of the form $\mu x.t$ denote least (pre-)fixed points. We show that under these interpretations the valid equations between μ-terms possess a finite axiomatization *over the axioms of iteration algebras* (or iteration theories) [8], which capture the equational properties of the least fixed point operation on (ω-)continuous or monotonic functions. We prove that the following set of equations is relatively complete, where for any μ-terms $t, t', t \leq t'$

* Partially supported by grant no. FKFP 247/1999 from the Ministry of Education of Hungary and grant no. T30511 from the National Foundation of Hungary for Scientific Research.

P. Clote and H. Schwichtenberg (Eds.): CSL 2000, LNCS 1862, pp. 302–316, 2000.

is an abbreviation for $t + t' = t'$.

$$x + (y + z) = (x + y) + z \tag{1}$$
$$x + y = y + x \tag{2}$$
$$x + 0 = x \tag{3}$$
$$\mu x.x = 0 \tag{4}$$
$$\sigma(x_1, \ldots, x_n) \leq \sigma(x_1 + y_1, \ldots, x_n + y_n), \quad \sigma \in \Sigma_n,\ n > 0 \tag{5}$$
$$\mu x.t \leq \mu x.t + t' \tag{6}$$
$$\mu x.x + y = y \tag{7}$$

(As usual, the scope of the prefix μx extends to the right as far as possible. The equation

$$x + x = x \tag{8}$$

is a consequence of the above axioms and those of iteration algebras. Note that (6) is an equation scheme. Equations (5) may be replaced by the equation scheme (14).) As a byproduct of our relative completeness result, we obtain complete infinite equational and finite implicational axiomatizations. In fact, it follows that the system consisting of the equations (1) – (7), the Conway equations [8,9]

$$\mu x.t[t'/x] = t[\mu x.t'[t/x]/x] \tag{9}$$
$$\mu x.t[x/y] = \mu x.\mu y.t, \tag{10}$$

and an equation associated with each finite (simple) group is complete. (Group-equations for μ-terms were introduced in [15] as a generalization of Conway's group-equations for regular languages, cf. [10]. The completeness of the Conway equations and the group-equations for iteration algebras extends Krob's result [21] who confirmed a longstanding conjecture of Conway [10] about the axiomatization of the equational theory of regular sets.) By recent advances in the study of iteration theories [14], it follows that the implicational system consisting of (1) – (3), (5), (8), the fixed point equation

$$\mu x.t = t[\mu x.t/x] \tag{11}$$

and the least pre-fixed point rule (or fixed point induction) [24,3]

$$t[y/x] \leq y \Rightarrow \mu x.t \leq y \tag{12}$$

is also complete. This result is analogous to Kozen's axiomatization [20] of the equational theory of the regular sets that improves on Salomaa's axiomatization [26], which is not a pure implicational system and is not sound in most of the natural models. We also show that there is no finite equational axiomatization and establish the existence of a polynomial time decision algorithm for the validity of equations. Along the way of proving these results, we give a concrete description of the free algebras in the corresponding variety of iteration algebras. This description uses simulation equivalence classes of regular

synchronization trees [25,23]. Thus, our axioms are also sound and complete for simulation equivalence of (regular) processes, connecting our work to a large body of axiomatization results in process algebra, of which [22,16,2,12] is only a small sampling.

2 The Models

2.1 Preiteration Algebras

Terms, or *μ-terms over a signature Σ* are defined by the syntax

$$T ::= x \mid \sigma(\overbrace{T, \ldots, T}^{n-\text{times}}) \mid \mu x.T,$$

where x ranges over a countably infinite set X of variables, and for each $n \geq 0$, σ ranges over Σ_n. Thus, the terms given in the Introduction may be regarded as μ-terms over the signature $\Sigma_{+,0}$ obtained by adding the binary letter $+$ and the constant 0 to Σ. A term with no occurrence of a prefix μx is called *finite*. *Free* and *bound variables* in a term are defined as usual. *We identify any two μ-terms that differ only in the names of the bound variables.* Moreover, for any μ-terms $t, t_1, \ldots, t_n$ and distinct variables $x_1, \ldots, x_n$, we write $t[t_1/x_1, \ldots, t_n/x_n]$ or $t[t_1, \ldots, t_n/x_1, \ldots, x_n]$ for the term that results from t by simultaneously substituting t_i for x_i, for each $i \in [n]$. Since we may assume that the bound variables in t are different from the variables that have a free occurrence in the terms t_i, no free variable in the t_i may become bound as the result of the substitution.

A *preiteration Σ-algebra* is a set A together with an assignment of a function $t_A : A^X \to A$ to each term t over Σ subject to the following rules:

1. For each variable x and $a \in A^X$, $x_A(a) = a(x)$, i.e., x_A is the projection $A^X \to A$ corresponding to x.
2. If $a, b \in A^X$ are such that $a(x) = b(x)$ for all variables x with a free occurrence in t, then $t_A(a) = t_A(b)$.
3. For all terms $t, t_1, \ldots, t_n$ and $a \in A^X$, $(t[t_1/x_1, \ldots, t_n/x_n])_A(a) = t_A(b)$, where $b(x_i) = (t_i)_A(a)$, $i \in [n]$, and $b(x) = a(x)$, if $x \notin \{x_1, \ldots, x_n\}$.
4. For all terms t, t' and variable x, if $t_A = t'_A$, then $(\mu x.t)_A = (\mu x.t')_A$.

When $a(x) = a_x \in A$, for each variable $x \in V$, where $V \subseteq X$ contains the free variables of t, below we will often write $t_A(a_x/x)$ or just $t(a_x/x)$ for $t_A(a)$. When t has no free variable, we also write t_A. A *homomorphism* $A \to B$ of preiteration Σ-algebras is a function $h : A \to B$ such that $t_A(a_x/x)h = t_B(a_x h/x)$, for all terms t and $a_x \in A$, $x \in X$. If A and B are preiteration Σ-algebras such that A is a subset of B and the inclusion of A into B is a homomorphism, we call A a *sub-preiteration* Σ-algebra of B. Moreover, if h is a surjective homomorphism $A \to B$, then we call B a *quotient* of A. In particular, if θ is a (preiteration Σ-algebra) *congruence* on A, i.e., θ is an equivalence relation on A such that $a(x)\,\theta\,b(x)$, $x \in X$ implies $t(a)\,\theta\,t(b)$, for all terms t and $a, b \in A^X$, and such that for all terms t, t' and if $t(a)\,\theta\,t'(a)$ holds for $a \in A^X$, then for any variable x,

$(\mu x.t)(a)\,\theta\,(\mu x.t')(a)$ for all $a \in A^X$, then the factor set A/θ can be turned into a preiteration Σ-algebra in a unique way such that the quotient map $A \to A/\theta$ becomes a homomorphism.

Example 1. Recall that an *ω-continuous Σ-algebra* is a Σ-algebra A which is an ω-complete poset such that the operations $\sigma_A : A^n \to A$ induced by the letters $\sigma \in \Sigma_n$, $n \geq 0$ are ω-continuous, i.e., they preserve the supremum of any ω-chain. Each ω-continuous Σ-algebra may be turned into a preiteration Σ-algebra such that if $t = \sigma(t_1, \ldots, t_n)$, for some $\sigma \in \Sigma_n$ and terms $t_1, \ldots, t_n$, then $t_A(a) = \sigma_A((t_1)_A(a), \ldots, (t_n)_A(a))$, for all $a \in A^X$. Moreover, if $t = \mu x.t'$, for some variable x and μ-term t', then for each $a \in A^X$, $(\mu x.t)_A(a)$ is the least (pre-)fixed point of the ω-continuous function $A \to A$, $b \mapsto t'_A(a^x_b)$, where $a^x_b \in A^X$ agrees with a except that it maps x to b. A homomorphism of continuous Σ-algebras is a Σ-algebra homomorphism which is an ω-continuous function. It follows that any homomorphism of ω-continuous Σ-algebras is a homomorphism of the corresponding preiteration Σ-algebras.

Example 2. The previous example can be generalized. Suppose that A is a small (skeletal) category that has an initial object and colimits of all ω-diagrams. Moreover, suppose that for each $\sigma \in \Sigma_n$. σ_A is an ω-continuous functor $A^n \to A$, so that σ_A preserves colimits of ω-diagrams. Then we may assign a functor $t_A : A^X \to A$ to each term t over Σ using *initial fixed points*. The set of objects of A thus becomes a preiteration Σ-algebra. See [8] for details.

A *semilattice Σ-algebra* $(A, \Sigma, +, 0)$ has both the structure of a Σ-algebra (A, Σ) and the structure $(A, +, 0)$ of a semilattice with zero. A homomorphism of semilattice Σ-algebras preserves the Σ-algebra operations, the semilattice operation $+$ and the constant 0. An *ordered semilattice Σ-algebra* is a semilattice Σ-algebra which satisfies equations (5), so that the operations are monotonic with respect to the *induced partial order* defined by $a \leq b$ iff $a + b = b$. Homomorphisms of ordered semilattice Σ-algebras are just semilattice Σ-algebra homomorphisms. Note that semilattice Σ-algebras form a variety of $\Sigma_{+,0}$-algebras. This variety is axiomatized by the equations (1) – (3) and (8). Ordered semilattice Σ-algebras form a subvariety of semilattice Σ-algebras.

An *ω-continuous semilattice Σ-algebra* $A = (A, \Sigma, +, 0)$ is both a semilattice with zero $(A, +, 0)$ and an ω-continuous Σ-algebra (A, Σ), *equipped with the induced partial order*. It follows that all countable suprema exist in A. Alternatively, an ω-continuous semilattice Σ-algebra is a Σ-algebra which, equipped with the induced partial order, is an ω-continuous Σ-algebra. Homomorphisms of ω-continuous semilattice Σ-algebras are both semilattice homomorphisms and ω-continuous Σ-algebra homomorphisms. Since in an ω-continuous semilattice Σ-algebra $+$ is just the supremum operation with respect to the semilattice order, it follows that this operation is also ω-continuous. Thus, any ω-continuous semilattice Σ-algebra is an ω-continuous $\Sigma_{+,0}$-algebra. Also, any ω-continuous semilattice Σ-algebra is an ordered semilattice Σ-algebra, and any homomorphism of ω-continuous semilattice Σ-algebras is an (ordered) semilattice Σ-algebra homomorphism. (For a different definition of continuous semilattices see [18].)

Suppose that t and t' are terms over Σ. We say that *equation* $t = t'$ *holds in the preiteration* Σ*-algebra* A, or that A *satisfies* $t = t'$, if for all $a \in A^X$, $t_A(a) = t'_A(a)$, i.e., when the functions t_A and t'_A are equal. More generally, we say that the implication $t_1 = t'_1 \wedge \ldots \wedge t_n = t'_n \Rightarrow t = t'$ holds in the preiteration Σ-algebra A, where t, t', t_i, t'_i are terms over Σ, if for all $a \in A^X$, if $(t_i)_A(a) = (t'_i)_A(a)$, for all $i \in [n]$, then $t_A(a) = t'_A(a)$.
A *variety of preiteration* Σ*-algebras* is a class $\mathcal{V}$ of preiteration Σ-algebras consisting of the models of some set E of equations between terms over Σ, i.e., such that a preiteration Σ-algebra A belongs to $\mathcal{V}$ iff A satisfies any equation in E. The set E is called an *equational basis*, or an *equational axiomatization of* $\mathcal{V}$. It follows that each class $\mathcal{K}$ of preiteration Σ-algebras is contained in a least variety $\mathcal{V}$, the class of all models of the equations that hold in every member of $\mathcal{K}$. $\mathcal{V}$ is called the *variety generated by* $\mathcal{K}$. See [8] for more on varieties of iteration algebras.

2.2 Iteration Algebras

Some nontrivial equations that hold in all ω-continuous Σ-algebras are (9), (10), (11) given above. To define the group-equations, we need to extend the μ-notation to term vectors $t = (t_1, \ldots, t_n)$ over Σ. Let $x = (x_1, \ldots, x_n)$ be a vector of distinct variables. When $n = 1$, $\mu x.t$ is just the term vector of dimension 1 whose unique component is $\mu x_1.t_1$. We identify any term vector of dimension 1 with its component. If $n > 1$, let $x' = (x_1, \ldots, x_{n-1})$, $t' = (t_1, \ldots, t_{n-1})$ and $s = t'[\mu x_n.t_n/x_n]$. (Substitution into a term vector is defined componentwise.) We define

$$\mu x.t = (\mu x'.s, (\mu x_n.t_n)[\mu x'.s/x']). \tag{13}$$

The definition is motivated by the Bekić-de Bakker–Scott rule [5,3]. It follows that for any ω-continuous Σ-algebra A, term vector $t = (t_1, \ldots, t_n)$ of dimension n, and for any $x = (x_1, \ldots, x_n)$ and $a \in A^X$, $(\mu x.t)_A(a)$ is the least pre-fixed point of the map $A^n \to A^n$, $b = (b_1, \ldots, b_n) \mapsto t_A(a^x_b)$, where of course $a^x_b(x_i) = b_i$, for all $i \in [n]$, and $a^x_b(x) = a(x)$, if $x \notin \{x_1, \ldots, x_n\}$.
Suppose now that G is a finite group of order n with multiplication denoted $\cdot$. Moreover, suppose that the elements of G are the integers in the set $[n]$. Given a vector $x = (x_1, \ldots, x_n)$ of distinct variables and an integer $i \in [n]$, define $i \cdot x = (x_{i\cdot 1}, \ldots, x_{i\cdot n})$. Thus, $i \cdot x$ is obtained by permuting the components of x according to the ith row of the multiplication table of G. The *group-equation associated with* G is

$$(\mu x.(t[1 \cdot x/x], \ldots, t[n \cdot x/x]))_1 = \mu y.t[y/x, \ldots, y/x],$$

where t is any μ-term over Σ, y is a variable, and where $(\mu x.(t[1 \cdot x/x], \ldots, t[n \cdot x/x]))_1$ is the first component of the term vector $\mu x.(t[1 \cdot x/x], \ldots, t[n \cdot x/x])$.
An *iteration* Σ*-algebra* [8,15] is a preiteration Σ-algebra satisfying (9), (10), as well as each group-equation. A homomorphism of iteration algebras is a preiteration algebra homomorphism. A *sub-iteration algebra* of an iteration algebra is just a sub-preiteration algebra.

Theorem 1. [15] *The ω-continuous Σ-algebras generate the variety of iteration Σ-algebras.*

Thus, an equation between terms over Σ holds in all iteration Σ-algebras iff it holds in all ω-continuous Σ-algebras.
The above definitions apply to the case that the signature is of the form $\Sigma_{+,0}$. In particular, any ω-continuous semilattice Σ-algebra is an iteration $\Sigma_{+,0}$-algebra. In [14] it is proved that any "ordered preiteration Σ-algebra" satisfying the fixed point equation and the least pre-fixed point rule is an iteration Σ-algebra. Using this result we have:

Proposition 1. *Any preiteration $\Sigma_{+,0}$-algebra satisfying (1) – (3), (8), the fixed point equation (11) and the least pre-fixed point rule (12) is an iteration $\Sigma_{+,0}$-algebra and satisfies (4), (6) and (7).*

For later use, we note

Lemma 1. *Suppose that A is a preiteration $\Sigma_{+,0}$-algebra satisfying (1) – (3), (5), (6), (8). Then for all terms t over $\Sigma_{+,0}$, A satisfies the equation*

$$t \leq t[x + y/x], \tag{14}$$

or equivalently, the implication $x \leq y \Rightarrow t \leq t[y/x]$.

3 The Completeness Results

In this section we give precise formulations of the main completeness results of the paper.

Theorem 2. *An equation between terms over $\Sigma_{+,0}$ holds in all ω-continuous semilattice Σ-algebras iff it holds in all iteration $\Sigma_{+,0}$-algebras satisfying equations (1) – (7).*

Corollary 1. *An equation between terms over $\Sigma_{+,0}$ holds in all ω-continuous semilattice Σ-algebras iff it holds in all preiteration $\Sigma_{+,0}$-algebras satisfying the Conway equations (9), (10), the group-equations associated with the finite groups, and equations (1) – (7).*

Corollary 2. *An equation between terms over $\Sigma_{+,0}$ holds in all ω-continuous semilattice Σ-algebras iff it holds in all preiteration $\Sigma_{+,0}$-algebras satisfying equations (1) – (3), (5), (8), (11), and the least pre-fixed point rule (12).*

Corollary 3. *An equation between terms over $\Sigma_{+,0}$ holds in all ω-continuous semilattice Σ-algebras iff it holds in all preiteration $\Sigma_{+,0}$-algebras satisfying equations (1) – (3), (5), (6), (8), (11), and the* least fixed point rule *(15)*

$$t[y/x] = y \Rightarrow \mu x.t \leq y,$$

where t is any term over $\Sigma_{+,0}$.

To prove Theorem 2, we first give a concrete description of the free algebras in the variety of iteration $\Sigma_{+,0}$-algebras satisfying (1) – (7). In Theorem 6, we show that the free algebra on a set A may be represented as the iteration $\Sigma_{+,0}$-algebra of simulation equivalence classes of regular (Σ, A)-labeled synchronization trees. We then show in Theorem 7 that each such free algebra can be embedded in an ω-continuous semilattice Σ-algebra. Theorem 2 is an immediate consequence of these facts.

4 Synchronization Trees

In this section we consider a generalization of the usual notion of synchronization trees [23]. Suppose that Σ is a signature and A is a set disjoint from Σ. We extend the rank function on Σ to $\Sigma \cup A$ by defining the rank of each letter in A to be 0. The resulting signature is denoted $\Sigma(A)$. A (Σ, A)*-labeled (synchronization) tree* is a countable rooted hyper-tree (V, E, r) equipped with a labeling function λ subject to certain conditions. Here, V denotes the set of vertices and E is the set of (hyper-)edges, so that each edge $e \in E$ has a source $v \in V$ and a target $(v_1, \ldots, v_n) \in V^n$, for some $n > 0$, called the *rank* of e. Accordingly, we write $e : v \to (v_1, \ldots, v_n)$, and call v and the v_i the *endpoints* of e. We require that the labeling function is compatible with the ranks, so that $e\lambda \in \Sigma(A)_n$ whenever e has rank $n \neq 1$, and $e\lambda \in \Sigma_0 \cup A \cup \Sigma_1$, if e has rank 1. Moreover, we require that the target of any edge labeled in $\Sigma_0 \cup A$ is a leaf. Each synchronization tree has an underlying directed graph defined in a straightforward way.
If $S = (V, E, r, \lambda)$ and $S' = (V', E', r', \lambda')$ are (Σ, A)-labeled trees, a *simulation* [25] $S \to S'$ is a relation $\rho : V \to V'$ such that the roots are related, i.e., $r \,\rho\, r'$, and for all $e : v \to (v_1, \ldots, v_n)$ in E and $v' \in V'$, if $v \,\rho\, v'$ then there is an edge $e' : v' \to (v'_1, \ldots, v'_n)$ with $e\lambda = e'\lambda'$ and $v_i \,\rho\, v'_i$, for all $i \in [n]$. A *bisimulation* $S \to S'$ is a simulation $\rho : S \to S'$ such that ρ^{-1}, the relational inverse of ρ is a simulation $S' \to S$. A *functional simulation* is a simulation which is a function. It is obvious that the composition of simulations is a simulation. Thus (Σ, A)-labeled synchronization trees and their (functional) simulations form a category.

Proposition 2. *If ρ is a simulation $S \to S'$, where $S = (V, E, r, \lambda)$ and $S' = (V', E', r', \lambda')$, then there is a functional simulation $\tau : S \to S'$ contained in ρ, i.e., such that $(v, (v\tau)) \in \rho$ for all $v \in V$.*

Thus, there is a simulation $S \to S'$ iff there is a functional simulation $S \to S'$. Suppose that $S = (V, E, r, \lambda)$ is a (Σ, A)-labeled tree and $v \in V$. Let V_v denote the set of all vertices accessible from v along a path in the underlying directed graph of S. Let E_v denote the set of all edges $e : u \to (u_1, \ldots, u_n)$ such that $u \in V_v$ (and hence $u_1, \ldots, u_n \in V_v$), and let λ_v be the restriction of λ to E_v. The resulting (Σ, A)-labeled synchronization tree (V_v, E_v, v, λ_v) is denoted S_v. We call S_v the *subtree of S rooted at v*. We let $(\Sigma, A)T$ denote the category of all (Σ, A)-trees and functional simulations. Note that the isomorphisms in $(\Sigma, A)T$ are the bijective functional simulations. Two subcategories of $(\Sigma, A)T$ are also of interest: the category $(\Sigma, A)F$ determined by the finite trees, and the

category $(\Sigma, A)R$ determined by the *regular* trees. We call a tree regular if it has, up to isomorphism, a finite number of subtrees. Moreover, we call a functional simulation $\tau : S \to S'$ *normal* if for all vertices v_1, v_2 of S, if S_{v_1} and S_{v_2} are isomorphic, then so are $S'_{v_1\tau}$ and $S'_{v_2\tau}$. The proof of the following proposition is omitted.

Proposition 3. *Suppose that S and S' are trees and ρ is a functional simulation $S \to S'$. Then there exists a normal functional simulation $\tau : S \to S'$.*

Proposition 4. [8] *The category $(\Sigma, A)T$ is countably cocomplete.*

Colimits can be constructed in the expected way. If $S_i = (V_i, E_i, r_i, \lambda_i)$, $i = 1, 2$ are trees, then the coproduct $S_1 + S_2$ is the disjoint union of the S_i with the distinguished vertices identified. The coproduct injections are the obvious embeddings. See [8] for a formal definition of $S_1 + S_2$. The empty coproduct, i.e., initial object is the tree 0 with a single vertex and no edges. In addition to coproducts, we will use colimits of ω-diagrams $(S_n, f_n)_{n\geq 0}$, where $S_n = (V_n, E_n, r_n, \lambda_n)$ and $f_n : S_n \to S_{n+1}$. The colimit $(h_n : S_n \to S)$ can be constructed at the level of sets.

From now on, we identify isomorphic trees, so that we may regard $(\Sigma, A)T$, the category of (Σ, A)-trees and functional simulations, as a small skeletal category. For each $\sigma \in \Sigma_n$, we define the functor $\sigma_{(\Sigma,A)T} : (\Sigma, A)T^n \to (\Sigma, A)T$ as follows. Given trees $S_1, \ldots, S_n$ with roots $r_1, \ldots, r_n$, respectively, $\sigma_{(\Sigma,A)T}(S_1, \ldots, S_n) = \sigma(S_1, \ldots, S_n)$ is the tree obtained from the S_i by taking their disjoint union and adding to this set a new vertex r and a new edge $e : r \to (r_1, \ldots, r_n)$ labeled σ. Vertex r is the new root. On morphisms, $\sigma_{(\Sigma,A)T}$ is defined in the expected way. When $n = 0$, $\sigma_{(\Sigma,A)T}$ is a tree with a single edge labeled σ. The following fact is clear.

Proposition 5. [8] *Each functor $\sigma_{(\Sigma,A)T}$ is ω-continuous.*

Thus, since the functor $+$ that forms binary coproducts is also ω-continuous, using Example 2 we have:

Proposition 6. [8] *The isomorphism classes of (Σ, A)-trees form an iteration $\Sigma_{+,0}$-algebra satisfying the equations (1) – (4) and (15)*

$$\mu x.\mu y.x + y + z = \mu x.x + z. \tag{15}$$

We let $(\Sigma, A)\mathbf{T}$ denote this iteration $\Sigma_{+,0}$-algebra. It is shown in [8] that the regular trees determine a sub-iteration $\Sigma_{+,0}$-algebra of $(\Sigma, A)\mathbf{T}$. We denote this algebra by $(\Sigma, A)\mathbf{R}$. The finite trees determine a $\Sigma_{+,0}$-algebra $(\Sigma, A)\mathbf{F}$. The following result gives an algebraic characterization of $(\Sigma, A)\mathbf{R}$. *Let us identify each letter $a \in A$ with the tree which has a unique vertex and a unique edge, which is labeled a.*

Theorem 3. [8] *$(\Sigma, A)\mathbf{R}$ is freely generated by the set A in the variety of iteration $\Sigma_{+,0}$-algebras satisfying equations (1) – (4) and (15).*

The meaning of this result is that for any iteration $\Sigma_{+,0}$-algebra satisfying (1) – (4) and (15), and for any function $h : A \to B$, there is a unique iteration $\Sigma_{+,0}$-algebra homomorphism $h^\sharp : (\Sigma, A)\mathbf{R} \to B$ extending h. There is a similar result for finite trees.

Theorem 4. [6] $(\Sigma, A)\mathbf{F}$ *is freely generated by* A *in the variety of* $\Sigma_{+,0}$*-algebras satisfying equations* (1) – (3).

4.1 The Simulation Preorder

We will consider a preorder on trees. Suppose that S and T are (Σ, A)-labeled trees. We write $S \leq T$ if there is a (functional) simulation $S \to T$. The equivalence relation induced by this preorder $\leq$ is denoted $\equiv$. Relation $\leq$ is called the *simulation preorder* and $\equiv$ the *simulation equivalence*.

Proposition 7. *For all* (Σ, A)*-labeled trees* S *and* S'*, we have* $S \leq S'$ *iff* $S' \equiv S + S'$.

Suppose now that for each variable x we are given trees S_x and R_x with $S_x \leq R_x$. Then for any term t, since $t_{(\Sigma,A)T}$ is a functor $(\Sigma, A)T^X \to (\Sigma, A)T$, we have that $t_{(\Sigma,A)T}(S_x/x) \leq t_{(\Sigma,A)T}(R_x/x)$.

Proposition 8. *For all* μ*-terms* t *over* $\Sigma_{+,0}$ *and for all families of* (Σ, A)*-labeled trees* $(S_x)_{x\in X}$ *and* $(R_x)_{x\in X}$ *with* $S_x \leq R_x$ *for* $x \in X$*, it holds that* $t_{(\Sigma,A)T}(S_x/x) \leq t_{(\Sigma,A)T}(R_x/x)$.

Proposition 9. *Suppose that* s *and* t *are* μ*-terms over* $\Sigma_{+,0}$*. If* $s_{(\Sigma,A)T}(R_y/y) \leq t_{(\Sigma,A)T}(R_y/y)$*, for all families* $(R_y)_{y\in X}$ *of* regular *trees in* $(\Sigma, A)R$*, then also*

$$(\mu x.s)_{(\Sigma,A)T}(R_y/y) \leq (\mu x.t)_{(\Sigma,A)T}(R_y/y),$$

for all families $R = (R_y)_{y\in X}$ *of regular trees in* $(\Sigma, A)R$*, and for all* x.

Proof. For a family $(R_y)_{y\in X}$ of regular trees, let F denote the functor $(\Sigma, A)T \to (\Sigma, A)T$, defined on objects by $S \mapsto s_{(\Sigma,A)}(R_S^x)$. On morphisms, F is defined in a similar way. Let G denote the corresponding functor using term t. Then $F^\dagger = (\mu x.s)_{(\Sigma,A)T}(R_y/y)$ is the colimit of the ω-diagram $F^n(i_0) : F^n(0) \to F^{n+1}(0)$, where 0 is the empty tree and i_0 denotes the unique functional simulation $0 \to F(0)$. Also, $G^\dagger = (\mu x.t)_{(\Sigma,A)T}(R_y/y)$ is the colimit of the ω-diagram $G^n(j_0) : G^n(0) \to G^{n+1}(0)$ defined in the same way. It is easy to see that each $F^n(i_0)$ and $G^n(j_0)$ is injective, so that we may as well assume that each $F^n(i_0)$ and $G^n(j_0)$ is an inclusion, and that $F^\dagger$ and $G^\dagger$ are the "unions" of the $F^n(0)$ and $G^n(0)$, respectively. Suppose that S is a finite tree with $S \leq F^\dagger$. Then there is some n such that $S \leq F^n(0)$. But it follows from our assumption on s and t that $F^n(0) \leq G^n(0)$, so that $S \leq G^n(0)$ and $S \leq G^\dagger$. Since $F^\dagger$ and $G^\dagger$ are regular, by Corollary 4, proved independently, we have $F^\dagger \leq G^\dagger$. □

By the previous facts, simulation equivalence is a congruence on $(\Sigma, A)\mathbf{R}$. Let $(\Sigma, A)\mathbf{SR}$ denote the quotient $(\Sigma, A)\mathbf{R}/\equiv$.

Proposition 10. $(\Sigma, A)\mathbf{SR}$ *is an iteration* $\Sigma_{+,0}$*-algebra satisfying (1) – (7).*

Proof. Since $(\Sigma, A)\mathbf{SR}$ is a quotient of $(\Sigma, A)\mathbf{R}$, by Theorem 3 $(\Sigma, A)\mathbf{SR}$ is an iteration $\Sigma_{+,0}$-algebra satisfying (1) – (4). It follows from Proposition 7 that $(\Sigma, A)\mathbf{SR}$ also satisfies (8) and that the relation $\leq$ is the partial order induced by the semilattice structure. By Proposition 8, also (5) holds. Thus, it remains to verify (6) and (7). Equation (6) follows from Exercise 5.21 in Chapter 8 of [8]. Equation 7 is obvious. □

Let $(\Sigma, A)\mathbf{SF}$ denote the subalgebra of $(\Sigma, A)\mathbf{SR}$ determined by the finite trees.

Theorem 5. [17] *For each set* A, $(\Sigma, A)\mathbf{SF}$ *is freely generated by* A *in the variety of ordered semilattice* Σ*-algebras. An equation between finite* $\Sigma_{+,0}$*-terms holds in all algebras of simulation equivalence classes of (finite)* (Σ, A)*-trees iff it holds in all ordered semilattice* Σ*-algebras.*

5 A Characterization of Simulation Equivalence Classes of Regular Trees

In this section we give an algebraic characterization of simulation equivalence classes of regular (Σ, A)-labeled trees. In the technical developments to follow, for any term t over $\Sigma_{+,0}$ and for any integer $k \geq 0$, we will use the abbreviation kt for the k-fold sum of t with itself (where we take advantage of the associativity of $+$). When $k = 0$, kt is just the term 0. Moreover, we will write ∞t for the term $\mu x.x + t$, where the variable x does not occur in t.

A (Σ, A)*-normal description of dimension* n *in the variables* $x_1, \ldots, x_n, y_1, \ldots, y_p$ is an ordered pair $D = (t, a)$, where $t = (t_1, \ldots, t_n)$ is an n-dimensional vector of terms over $\Sigma_{+,0}$ in the free variables $x_1, \ldots, x_n, y_1, \ldots, y_p$ and $a = (a_1, \ldots, a_p) \in A^p$. Moreover, each term t_i is *primitive*, i.e., a finite sum of terms of the form $k\sigma(x_{j_1}, \ldots, x_{j_m})$ or ky_j, where $k \neq 0$, $\sigma \in \Sigma_m$, $m \geq 0$, $j_1, \ldots, j_m \in [n]$ and $j \in [p]$. (It is allowed that $k = \infty$.) Let us denote $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_p)$. The *behavior* of D, denoted $|D|$ is the first component of $(\mu x.t)_{(\Sigma,A)\mathbf{R}}(a/y)$. Thus, the behavior of D is a regular tree in $(\Sigma, A)\mathbf{R}$.

Each regular tree $T \in (\Sigma, A)\mathbf{R}$ is known to be the behavior of a description $D = (t, a)$. To construct D, let $T_1, \ldots, T_n$ be an enumeration of the subtrees of T with $T = T_1$, and let $a_1, \ldots, a_p$ be an enumeration of those elements of A which appear as labels of some edges of T. We define $t = (t_1, \ldots, t_n)$ and $a = (a_1, \ldots, a_p)$, where each t_i corresponds to T_i in the following manner. Each edge $e : v \to (v_1, \ldots, v_m)$ whose source is the root of T_i is labeled by some symbol $\sigma \in \Sigma_m$ or some component of a. Suppose that e is labeled $\sigma \in \Sigma_m$, $m > 0$. Then let $T_{j_1}, \ldots, T_{j_m}$ denote the subtrees rooted at the vertices $v_1, \ldots, v_m$, respectively, and let k denote the total number of edges $v \to (v'_1, \ldots, v'_m)$ labeled σ such that the subtrees rooted at the vertices $v'_1, \ldots, v'_m$ are isomorphic to the trees $T_{j_1}, \ldots, T_{j_m}$, respectively. (If there are an infinite number of such edges, then $k = \infty$.) Then $k\sigma(x_{j_1}, \ldots, x_{j_m})$ is a summand of t_i. Similarly, if e is labeled a_j, say, then ky_j is a summand of t_i, where k is determined in the same way.

Finally, if v has an outedge labeled $\sigma \in \Sigma_0$, then $k\sigma$ is a summand of t_i, where k is the number of all such edges. The term t_i is the sum of all such summands. We call the description D constructed in this way the *canonical description of* T. (Note that the canonical description is unique only up to a rearrangement of the components of t and a and renaming of the variables.) It is known that for each i, the ith component of $(\mu x.t)(a)$ is the tree T_i. In particular, we have:

Proposition 11. [8] *For the canonical description* $D = (t, a)$ *of the tree* $T \in (\Sigma, A)\mathbf{R}$*, it holds that* $|D| = T$.

Theorem 6. *For each set* A, $(\Sigma, A)\mathbf{SR}$ *is freely generated by* A *in the variety of iteration* (Σ, A)*-algebras satisfying equations (1) – (7).*

Proof. By Proposition 10, $(\Sigma, A)\mathbf{SR}$ is an iteration $\Sigma_{+,0}$-algebra satisfying (1) – (7). Suppose that B is an iteration (Σ, A)-algebra which also satisfies equations (1) – (7), and suppose that h is a function $A \to B$. By Theorem 3, h extends to a unique homomorphism $(\Sigma, A)\mathbf{R} \to B$ that we also denote by h. If we can show that $T \leq T'$ implies $Th \leq T'h$, for all trees $T, T' \in (\Sigma, A)\mathbf{R}$, then it follows that h factors through the quotient map $(\Sigma, A)\mathbf{R} \to (\Sigma, A)\mathbf{SR}$. Thus,

$$h = (\Sigma, A)\mathbf{R} \xrightarrow{\tau} (\Sigma, A)\mathbf{SR} \xrightarrow{h^\sharp} B,$$

where τ is the quotient map. Using Theorem 3, it follows that $h^\sharp$ is the unique extension of h to a homomorphism $(\Sigma, A)\mathbf{SR} \to B$.
So assume that T, T' are regular (Σ, A)-labeled trees with $T \leq T'$. Let $D = (t, a)$ and $D' = (t', a')$ denote the canonical descriptions of T and T', respectively, where $t = (t_1, \ldots, t_n)$, $t' = (t'_1, \ldots, t'_{n'})$, $a = (a_1, \ldots, a_p)$, $a' = (a'_1, \ldots, a'_{p'})$. Let $x_1, \ldots, x_n, y_1, \ldots, y_p$ and $x'_1, \ldots, x'_{n'}, y'_1, \ldots, y'_{p'}$ denote the free variables appearing in t and t', respectively, where each x_i corresponds to t_i, each y_j to a_j, etc. Recall that each x_i also corresponds to a subtree T_i of T, and similarly, each x'_i to a subtree T'_i of T'. Since $T \leq T'$, we have $p \leq p'$ and $\{a_1, \ldots, a_p\} \subseteq \{a'_1, \ldots, a'_{p'}\}$, so that without loss of generality we may assume that $a'_j = a_j$, for all $j \in [p]$. Let φ denote a functional simulation $T \leq T'$. By Proposition 3, we may assume that φ is normal. Thus, we can use φ to define a map $\psi : [n] \to [n']$ such that $T_i \leq T'_{i\psi}$, for each $i \in [n]$: if v is any vertex of T such that the subtree of T rooted at v is isomorphic to T_i, then we let $i\psi$ be the integer in $[n']$ such that the subtree of T' rooted at $v\varphi$ is isomorphic to $T'_{i\psi}$. For each $i \in [n]$, let $t_i\psi$ denote the term that results from t_i by substituting $x'_{j\psi}$ for x_j, $j \in [n]$, and y'_j for y_j, $j \in [p]$, i.e., $t_i\psi = t_i[(x'_{1\psi}, \ldots, x'_{n\psi}, y'_1, \ldots, y'_p)/(x_1, \ldots, x_n, y_1, \ldots, y_p)]$. Since φ is a simulation, whenever $k\sigma(x_{j_1}, \ldots, x_{j_m})$ is a summand of t_i, where $k \neq 0$, there exists some $k' \neq 0$ such that $k'\sigma(x'_{j_1\psi}, \ldots, x'_{j_m\psi})$ is a summand of $t'_{i\psi}$. Also, if for some $k \neq 0$, ky_j is a summand of t_i, then there is some $k' \neq 0$ such that $k'y'_j$ is a summand of $t'_{i\psi}$. It follows that with respect to equations (1) – (3), (8) and (7), we have that $t_i\psi + t'_{i\psi} = t'_{i\psi}$, i.e., $t_i\psi \leq t'_{i\psi}$. Thus for each i there is some primitive term s_i in the variables $x_1, \ldots, x_n, y_1, \ldots, y_p$ such that, modulo the equations (1) – (3), (8) and (7), we have

$$(t_i + s_i)\psi = t'_{i\psi}, \tag{16}$$

where $(t_i + s_i)\psi$ is defined in the same way as $t_i\psi$. Let $r = (r_1, \ldots, r_n)$, where $r_i = t_i + s_i$, for all $i \in [n]$, and let R denote the first component of $(\mu x.r)(a)$. We use equations (16) to prove

Lemma 2. *There is a bisimulation $R \to T'$.*

It was shown in [8] that regular trees in $(\Sigma, A)\mathbf{R}$ modulo bisimulation form an iteration algebra $(\Sigma, A)\mathbf{BR}$ freely generated by A in the variety of iteration $\Sigma_{+,0}$-algebras satisfying (1) – (4) and (7). Thus, by Lemma 2 we have that $Rh = T'h$. On the other hand, by Lemma 3, $Th \leq Rh$. Thus, $Th \leq T'h$. □

In the next lemma we will say that an inequation $(t_1, \ldots, t_n) \leq (s_1, \ldots, s_n)$ between vectors of terms over $\Sigma_{+,0}$ holds in a preiteration $\Sigma_{+,0}$-algebra, or in a class of preiteration $\Sigma_{+,0}$-algebras, if each equation $t_i + s_i = s_i$ holds.

Lemma 3. *Suppose that $t = (t_1, \ldots, t_n)$ and $s = (s_1, \ldots, s_n)$ are term vectors in the free variables $x_1, \ldots, x_n, y_1, \ldots, y_p$. Let $x = (x_1, \ldots, x_n)$. If $t \leq s$ holds in the variety $\mathcal{V}$ of iteration $\Sigma_{+,0}$-algebras satisfying (1) – (7), then so does $\mu x.t \leq \mu x.s$.*

Proof. By induction on n using (13). The basis case holds by (6). □

6 An Embedding Theorem

In this section we prove that for each set A, there is an ω-continuous semilattice Σ-algebra B such that $(\Sigma, A)\mathbf{SR}$ can be embedded in B. We call a tree $T \in (\Sigma, A)\mathbf{T}$ *finitely branching* if each vertex of T is the source of a finite number of hyper-edges. For a tree $T \in (\Sigma, A)\mathbf{T}$, we denote by $K(T)$ the collection of all finite trees S with $S \leq T$.

Lemma 4. *Suppose that $T, T' \in (\Sigma, A)\mathbf{T}$ such that T' is finitely branching. Then $T \leq T'$ iff $K(T) \subseteq K(T')$.*

Proof. It is clear that $K(T) \subseteq K(T')$ whenever $T \leq T'$. The reverse implication is obvious when T is finite. So assume that T is infinite, say $T = (V, E, r, \lambda)$. Let $\overline{K}(T)$ denote the collection of all trees $S = (W, F, r, \lambda_F)$ such that W and F are finite subsets of V and E, respectively, and the inclusion $W \to V$ determines a functional simulation $S \to T$. Then let $T_0 = (V_0, E_0, r, \lambda_0), T_1 = (V_1, E_1, r, \lambda_1), \ldots$ be a sequence of trees in $\overline{K}(T)$ such that each vertex $v \in V$ appears in all but a finite number of the V_i. Since $\overline{K}(T) \subseteq K(T) \subseteq K(T')$, for each i there exists a functional simulation $\rho_i : T_i \to T'$. We show how to construct a functional simulation $\rho : T \to T'$. Let v denote a vertex in V. We define $v\rho$ by induction on the depth of v such that $v \in V_i$ and $v\rho = v\rho_i$ hold for an infinite number of the i's. When v is the root r, we define $v\rho = r'$, the root of T'. Suppose now that the depth of v is positive. Let u denote the vertex such that there is an edge $u \to (v_1, \ldots, v_m)$ in E with $v_j = v$ for some j. By the induction assumption, there is an infinite set I such that $u \in V_i$ and $u\rho = u\rho_i$, for all $i \in I$. Moreover, by our assumption on the sequence T_k, $k \geq 0$, there is an infinite set $I' \subseteq I$ such that v is a vertex of T_i, for all $i \in I'$, and since T'

is finitely branching, there is an infinite set $I'' \subseteq I'$ such that $v\rho_i$ is the same vertex of T', for all $i \in I''$. Define $v\rho$ to be this vertex of T'. □

Given an edge $e : v \to (v_1, \ldots, v_n)$ of a tree $S = (V, E, r, \lambda)$, the vertices of the tree T determined by e are the vertex v and those vertices $v' \in V$ accessible from the v_i in the underlying directed graph of S. The root of T is v, and its edges are those edges in E whose endpoints belong to the vertex set of T. The labeling function is the restriction of the labeling function λ. We call a (Σ, A)-labeled tree S *reduced* if for any two distinct edges $e : v \to (v_1, \ldots, v_n)$ and $e' : v \to (v'_1, \ldots, v'_n)$ with the same source, the trees determined by e and e' are incomparable with respect to $\leq$, i.e., either e and e' have distinct labels or there exists some $i \in [n]$ such that $S_{v_i} \not\leq S_{v'_i}$ and $S_{v'_i} \not\leq S_{v_i}$.

Lemma 5. *Suppose that S and S' are reduced (Σ, A)-labeled trees. Then $S \equiv S'$ iff S and S' are isomorphic.*

Proposition 12. *If S is a regular (finite, respectively) (Σ, A)-labeled tree, then up to isomorphism there is a unique reduced tree S' with $S \equiv S'$. Moreover, S' is also regular (finite, respectively).*

Example 3. Suppose that $\sigma \in \Sigma_1$. Define $\sigma^0(0) = 0$ and $\sigma^{n+1}(0) = \sigma(\sigma^n(0))$, so that each $\sigma^n(0)$ is reduced. Let $T = \sum_{n\geq 0} T_n$. Then there is no reduced tree simulation equivalent to T. On the other hand, if $S_1, \ldots, S_k$ are reduced, then there is a reduced tree simulation equivalent to $S_1 + \ldots + S_k$.

By the above results, we may represent $(\Sigma, A)\mathbf{SF}$ as an ordered semilattice $\Sigma_{+,0}$-algebra of finite reduced (Σ, A)-labeled trees. In the same way, $(\Sigma, A)\mathbf{SR}$ may be represented as an iteration $\Sigma_{+,0}$-algebra of reduced regular trees. Since any reduced regular tree is finitely branching, from Lemma 4 and Proposition 12 we deduce

Corollary 4. *Suppose that T, T' are (Σ, A)-labeled trees and T' is regular. Then $T \leq T'$ iff $K(T) \subseteq K(T')$.*

Example 4. There exist nonisomorphic trees T, S such $K(T) = K(S)$. Indeed, let $T = \sum_{n\geq 0} \sigma^n(0)$ and $S = \sigma^\omega$, i.e., the colimit of the ω-diagram $(\sigma^n(0) \to \sigma^{n+1}(0))_{n\geq 0}$.

Theorem 7. *For each set A there exists an ω-continuous semilattice Σ-algebra B and an injective iteration $\Sigma_{+,0}$-algebra homomorphism $(\Sigma, A)\mathbf{SR} \to B$.*

Proof. By Theorem 5, the algebra $(\Sigma, A)\mathbf{SF}$ of simulation equivalence classes of finite (Σ, A)-labeled trees is freely generated by A in the variety of ordered semilattice Σ-algebras. Now $(\Sigma, A)\mathbf{SF}$ is a "strict ordered algebra", so that its completion $B = (\Sigma, A)\mathbf{ISF}$ by "ω-ideals" is a ω-continuous $\Sigma_{+,0}$-algebra satisfying every equation satisfied by $(\Sigma, A)\mathbf{SF}$. In particular, $(\Sigma, A)\mathbf{ISF}$ is an ω-continuous semilattice Σ-algebra. (In fact, it follows from well-known facts that $(\Sigma, A)\mathbf{ISF}$ is the free ω-continuous semilattice Σ-algebra on A, see [7].) By Corollary 4, the map $T/\equiv \mapsto K(T)$, $T \in (\Sigma, A)\mathbf{R}$ is a well-defined injective function $(\Sigma, A)\mathbf{SR} \to B$. We show that this function is an iteration $\Sigma_{+,0}$-algebra homomorphism. □

7 Further Results

By the Knaster-Tarski theorem, every monotonic function on a complete lattice has a least fixed point. Accordingly, one might wish to consider algebras over complete lattices equipped with monotonic or continuous operations, the binary supremum operation and the constant 0 representing the least element. The axioms of Theorem 2 and Corollaries 1, 2, 3 are sound in these models. Moreover, every iteration algebra $(\Sigma, A)\mathbf{SR}$ can be embedded in an iteration algebra derived from a complete lattice equipped with continuous operations. Hence, our main results remain valid for these models as well. Also, the least fixed point operation may be replaced by the greatest fixed point operation provided the binary supremum operation is replaced by the infimum operation. Very little is known about the equational theory when infimum and supremum and both extremal fixed point operations are present.

It is well-known that the problem to decide whether two finite transition systems are bisimilar lies in **P** (see, e.g, [19]), in fact, it is **P**-complete, cf. [4]. It is not difficult to show that the same holds for simulation equivalence. Now each μ-term t over $\Sigma_{+,0}$ can be transformed in logarithmic space to a finite transition system $S(t)$ such that $t = t'$ holds in ω-continuous Σ-algebras for terms t, t' iff $S(t)$ is simulation equivalent to $S(t')$ (i.e., when the tree obtained by unfolding $S(t)$ is simulation equivalent to the tree obtained by unfolding $S(t')$.) In conclusion, there is a polynomial algorithm to decide whether an equation holds in ω-continuous semilattice Σ-algebras. Suppose that Σ contains a letter whose rank is not 0. Then iteration Σ-algebras do not possess a finite axiomatization in terms of equation schemes [9]. It is not difficult to modify the proof of this result to show that the variety generated by ω-continuous semilattice Σ-algebras also do not possess a finite axiomatization. This also follows from the fact that Kleene algebras of binary relations (or regular languages) have no finite basis of their equations, see [10] or [2] for a recent improvement, but have a finite basis over the equations of iteration algebras [8], and in fact relative to the equations of ω-continuous semilattice Σ-algebras.

As for related results to be published in a forthcoming paper, we would like to mention a similar treatment of *continuous additive Σ-algebras*. The equational theory of these algebras is intimately related to that of the *resource simulation equivalence* classes of processes, or trees [11,13].

References

1. L. Aceto, W.J. Fokkink and A. Ingólfsdóttir, A menagerie of non-finitely based process semantics over BPA*: from ready simulation to completed traces, *Math. Struct. Comput. Sci.,* 8(1998), 193-230.
2. L. Aceto, W.J. Fokkink and A. Ingólfsdóttir, On a question of A. Salomaa: The equational theory of regular expressions over a singleton alphabet is not finitely based, *Theoret. Comput. Sci.*, 209(1998), 163-178.
3. J.W. De Bakker and D. Scott, A theory of programs, IBM Seminar, Vienna, 1969.
4. J. Balcazar, J. Gabarro and M. Santha, Deciding bisimilarity is P-complete, *Formal Aspects of Computing*, 4(1992), 638–648.

5. H. Bekić, Definable operations in general algebras, and the theory of automata and flowcharts, *Technical Report*, IBM Laboratory, Vienna, 1969.
6. D. Benson and J. Tiuryn, Fixed points in free process algebras, *Theoret. Comput. Sci.*, 63(1989), 275–294.
7. S.L. Bloom, Varieties of ordered algebras, *J. Comput. System Sci.*, 13(1976), 200–212.
8. S.L. Bloom and Z. Ésik, *Iteration Theories*, Springer–Verlag, 1993.
9. S.L. Bloom and Z. Ésik, The equational logic of fixed points, *Theoret. Comput. Sci.*, 179(1997), 1–60.
10. J.H. Conway, *Regular Algebra and Finite Machines*, Chapman and Hall, London, 1971.
11. F. Corradini, R. De Nicola and A. Labella, Tree morphisms and bisimulations, in: Proc. *MFCS'98 Workshop on Concurrency, ENTCS*, 18(1998).
12. F. Corradini, R. De Nicola and A. Labella, A finite axiomatization of nondeterministic regular expressions, *Theoret. Inform. Appl.*, 33(1999), 447–465.
13. F. Corradini, R. De Nicola and A. Labella, Models of nondeterministic regular expressions. *J. Comput. Sys. Sci.*, 59:412–449, 1999.
14. Z. Ésik, Completeness of Park induction, *Theoret. Comput. Sci.*, 177(1997), 217–283.
15. Z. Ésik, Group axioms for iteration, *Inform. and Comput.*, 148(1999), 131–180.
16. W. Fokkink and H. Zantema, Basic process algebra with iteration: Completeness of its equational axioms. *Computer Journal*, 37(1994), 259–267.
17. R.J.H. van Glabbeek, The linear time – branching time spectrum, Chapter 1 in: *Comparative Concurrency Semantics and Refinement of Actions*, R.J.H. van Glabbeek, CWI TRACT 109, 1996.
18. C.C. Gunter, *Semantics of Programming Languages*, MIT Press, 1992.
19. P.C. Kanellakis and S.A. Smolka, CCS expressions, finite state processes and three problems of equivalence, *Inform. and Comput.*, 86(1990), 43–68.
20. D. Kozen, A completeness theorem for Kleene algebras and the algebra of regular events, *Inform. and Comput.*, 110(1994), 366–390.
21. D. Krob, Complete systems of B-rational identities, *Theoret. Comput. Sci.*, 89(1991), 207–343.
22. R. Milner, A complete inference system for a class of regular behaviours, *J. Comput. Syst. Sci.*, 28(1984), 439–466.
23. R. Milner, *Communication and Concurrency*, Prentice-Hall, 1989.
24. D.M.R. Park, Fixpoint induction and proofs of program properties, in: *Machine Intelligence* **5**, D. Michie and B. Meltzer, Eds., Edinburgh Univ. Press, 1970, 59–78.
25. D.M.R. Park, Concurrency and automata on infinite sequences, in: *Proc. GI Conference*, P. Deussen, Ed., LNCS 104, Springer–Verlag, 1981, 167–183.
26. A. Salomaa, Two complete axiom systems for the algebra of regular events. *J. Assoc. Comput. Mach.*, 13(1966), 158–169.

Interactive Programs in Dependent Type Theory

Peter Hancock[1] and Anton Setzer[2]

[1] Dept. of Computing Science, University of Edinburgh, James Clerk Maxwell Building, King's Buildings, Mayfield Road, Edinburgh EH9 3JZ, Scotland, fax: +44 131 667 7209, phone: +44 131 650 5129, pgh@dcs.ed.ac.uk.

[2] Dept. of Mathematics, Uppsala University, P.O. Box 480, SE-751 06 Uppsala, Sweden, fax: +46 18 4713201, phone: +46 18 4713284, setzer@math.uu.se.

Abstract. We propose a representation of interactive systems in dependent type theory. This is meant as a basis for an execution environment for dependently typed programs, and for reasoning about their construction. The inspiration is the 'I/O-monad' of Haskell. The fundamental notion is an I/O-tree; its definition is parameterised over a general notion of dependently typed, command-response interactions called a world. I/O-trees represent strategies for one of the parties in a command/response interaction – the notion is not confined to functional programming. We present I/O-trees in two forms. The first form, which is simpler, is suitable for Turing-complete functional programming languages with general recursion, but is non-normalising. The second is definable within (ordinary) normalising type theory and we identify programs written in it as 'normalising I/O-programs'. We define new looping constructs (while and repeat), and a new refinement construct (redirect), which permits the implementation of libraries. We introduce a bisimulation relation between interactive programs, with respect to which we prove the monad laws and defining equations of while. Most definitions in this article make essential use of the expressive strength of dependent typing.

Keywords. Functional programming, reactive programming, interaction, dependent types, monadic I/O, repetition constructs, refinement.

1 I/O Concepts in Type Theory

Programming languages based on dependent types. Some 20 years ago, Martin-Löf [6] suggested that his type theory, originally a framework for constructive mathematics, could be considered as a programming language, and his suggestion has been taken up and explored in a number of ways (see for example [9]). A question which seems to have received little attention is the form of the input-output interface of such programs. Indeed it is only in the last 10 years that this question has been satisfactory answered in the context of conventional functional programming, through the efforts of Moggi [8], Wadler [13], and others.

Dependent types give us the ability to express with full precision any extensional property of a program, which can be defined mathematically. For example, we can express the requirement for a function which maps lists to sorted lists

P. Clote and H. Schwichtenberg (Eds.): CSL 2000, LNCS 1862, pp. 317–331, 2000.

using a dependent type [9]. Remarkably, with certain provisos of largely academic interest, we can still check the type of a program mechanically, and type-correctness carries full assurance that it satisfies its specification. In the past few years, implementations of dependent type systems for functional programming have begun to appear [1]. So far however the implications of dependent types for specification of interfaces and programs have not been examined.

Conventions, and plan of paper. In the following, we will work in a standard dependent type theory (for example [9]) with the usual introduction and elimination rules, including intensional equality, extended by some other rules. We will refer to it simply as 'type theory'. The notation we use, which is for the most part standard, is summarised in the appendix of the paper. Note that we sometimes omit indices and superscripts.

The plan of the paper is as follows. In the remainder of this section, we explain why one needs a model of interaction in type theory, and recall the approach taken in the functional programming language Haskell, using a monadic type form whose values are I/O-programs. In the next section, we present our extension of this notion, making use of type dependency. The third section introduces two repetition constructs (`while` and `repeat`), and a refinement or redirection construction. In the fourth, we point out that the repetition constructs can destroy normalisation, and develop an alternative formulation, which preserves it. Finally there is a concluding section, followed by a summary of our notation.

The need for interactive programs in type theory. Traditional 'batch' programs may be written in type theory as functions from input values (given in advance) to output values. The output from such a program is the result of applying the function to its input. This batch model is adequate for a large class of programs, typically numerical search or optimisation programs. It is not however adequate for a program which runs, say, in the guidance system of an airplane.

The programs one is ordinarily confronted with interact with their environment while they are running. We give input via devices like keyboard or microphone and get output via devices like monitor or loudspeaker, and this input-output cycle is repeated again and again. Programs may also interact with the file system, the network or via physical sensors and actuators of some kind. So if we want to use type theory as a practical functional programming language, we have to consider how to use it to write interactive (or reactive) programs.

Some approaches to interactive programs in type theory. In conventional functional programming, several approaches to interaction have been pursued. A good survey of some of these approaches is made in [10]: dialogues (or lazy streams), continuations, and monadic I/O. Mention should also be made of the 'uniqueness types' of the language Concurrent Clean[1]. In this paper we follow the monadic approach, introduced by E. Moggi [8], upon which the input/output (I/O-) system of the language Haskell[2] has been erected. A monad (the concept comes from category theory) is a triple $(\mathrm{M}, *, \eta)$, whose components, written in

[1] `http://www.cs.kun.nl/~clean`
[2] `http://haskell.org`

dependent type theory, have types

$$\begin{aligned} &\mathrm{M} : \mathrm{Set} \to \mathrm{Set} \ , \\ &* : (A, B : \mathrm{Set}, p : \mathrm{M}\,A, q : A \to \mathrm{M}\,B) \to \mathrm{M}\,B \ , \\ &\eta : (A : \mathrm{Set}, a : A) \to \mathrm{M}\,A \ , \end{aligned}$$

such that the following laws hold with respect to a given equivalence relation $=$. (Instead of $\eta\,A\,a$ we will write η_a^A).

$$\begin{aligned} \eta_a^A *_{A,B} q &= q\,a \ , \\ p *_{A,B} \lambda x.\eta_x^A &= p \ , \\ (p *_{A,B} q) *_{B,C} r &= p *_{A,B} (\lambda x.q\,x *_{B,C} r) \ . \end{aligned}$$

A special case of a monad is the I/O-monad. When referring to the I/O-monad, we write $(\mathrm{IO}\,A)$ instead of $(\mathrm{M}\,A)$. The interpretation of IO is as follows.

(a) For a given set A, $(\mathrm{IO}\,A)$ is the set of interactive programs that may or may not terminate, but terminate only with a result a of type A.
(b) The program $p * q$ first executes p. If p terminates with result a, then the program continues with $(q\,a)$. The result of the whole program is the result of $(q\,a)$.
(c) The program η_a simply terminates with result a, without any interaction.

Additionally, one adds functions for specific interactions. For example, we can deal with programs that communicate by writing and reading strings (such as text lines): $\mathsf{write} : \mathsf{String} \to \mathrm{IO}\,\mathbf{1}$, $\mathsf{read} : \mathrm{IO}\,\mathsf{String}$. Here $(\mathsf{write}\,s)$ is the program that outputs s on some device and returns $\bullet$, and $\mathsf{read} : \mathrm{IO}\,\mathsf{String}$ is the program that reads a string and returns it.

For the reader unfamiliar with type checking programs written using an I/O-monad, it is worth stressing that type checking interactive programs does not require itself any interaction, since we type check the program*text*.

For the I/O-monad, one sees that the laws mentioned above should hold with respect to an equality that identifies behaviourally equivalent programs.

Interactive programs are written in Haskell by using a form of the I/O-monad that gives access to the usual facilities of an operating system, including files, graphics, and time, as well as control features like exception handling or multi-threading.

The I/O-monad seems to be the most promising approach for the representation of interactive programs within dependent type theory. To add it as a new concept would however involve adding besides new typing judgements also new judgement level equations for the monad laws. This is more than the relativisation of type theory to a context of typed variables. The implications of this for the metamathematical properties of type theory are unclear to the authors.

Since we have access to powerful data type constructions in type theory, an easier approach is to define the I/O-monad and derive the monad laws directly in type theory. We still need something beyond mere evaluation of expressions, namely the ability to actually run an I/O-program. However we can use elimination rules for modifying I/O-programs and we shall make substantial use of

it in the following. Note that for efficiency reasons one might implement the execution of I/O programs in a different way, using for example a continuation monad with an 'answer' type of I/O programs. The paper [4] describes some of the options one has for implementing datatypes such as monads.

2 I/O-Trees

Worlds. Interactive programs are built from interactive commands. In dependent type theory we can define such sets of interactive commands in a very general way, parameterise over them and switch between command sets.

Let C be a set of instructions or commands. These include commands to obtain input, commands to produce output, and commands with a mixed effect. For commands $c : C$ let $(R\,c)$ be a type of responses produced when command c has been performed. $\langle C, R\rangle$ will be called a world:

General assumption and definition 2.1. *A* world w *is a pair* $\langle C, R\rangle$ *such that* $C : \mathrm{Set}$ *and* $R : C \to \mathrm{Set}$. *In the following* w *is always a world* $\langle C, R\rangle$. *We will in most cases omit the parameter* w.

Examples for constructors of C might be

(a) $\mathsf{write} : \mathsf{String} \to C$ with $R(\mathsf{write}\,s) = \mathbf{1}$: $\mathsf{write}\,s$ is the command for writing s and returning $\bullet : \mathbf{1}$ for success.
(b) $\mathsf{read} : C$, with $(R\,\mathsf{read} = \mathsf{String})$: read a string and return it.

Of course in practice the commands would be more complex. For example, there might be an embellishment of write where $R(\mathsf{write}\,s) = \{\mathsf{success}, \mathsf{fail}\}$ and $(\mathsf{write}\,r)$ returns the information whether the output was performed successfully or not. We might as well have commands for interaction with file systems, network etc.

I/O-trees. We want to define the I/O-monad as a data type constructed in type theory. It seems particularly suitable to define it as an inductive data type, because we can then carry out program transformations using the elimination rule associated with such types. A naïve idea would be to take $*$, η and the additional primitive instructions such as read and $(\mathsf{write}\,a)$ as constructors for this type. However we need to verify the monad laws, and it turns out that the naïve approach would require us to define a rather complicated equality relation. The situation is analogous to the definition of the set of natural numbers. We could define it from 0, 1 and $+$, which correspond in the I/O-monad to η, the primitive instructions and $*$. It is however much better to take 0 and successor S as constructors, and to define 1 and addition. What corresponds now to S in the IO-monad? This should be the operation which takes an instruction and a family or 'jump table' of programs depending on the result of performing that instruction, and creates a new program that begins by issuing this instruction and then, when the instruction has been performed, continues with the program determined by its result. Instructions are given by C, where $w = \langle C, R\rangle$ is a

world, and R provides the result type. So we have the following rules for $\mathrm{IO}_w\, A$:

$$\begin{aligned}&\mathrm{IO}_w : \mathrm{Set} \to \mathrm{Set}, \textbf{ where } \mathrm{IO}_w\, A \textbf{ has constructors}\\ &\mathsf{leaf} : A \to \mathrm{IO}_w\, A\ ,\\ &\mathsf{do} : (c : C, p : R\, c \to \mathrm{IO}\, A) \to \mathrm{IO}\, A\ .\end{aligned}$$

The constructor $(\mathsf{leaf}\, a)$ is what was written $\eta_A a$ before, and $(\mathsf{do}\, c\, p)$ denotes the program that first issues the command c, and depending on the result $r : R\, c$ returned by the environment continues with $(p\, r)$. Note that $(\mathrm{IO}_w\, A)$ is now parameterised with respect to w, a feature expressible only with dependent types.

$(\mathrm{IO}_w\, A)$ is the set of well-founded I/O-trees with leaves in A and inner nodes labelled by some $c : C$ and with branching degree $(R\, c)$ (ie. the subtrees of that node are indexed over $(R\, c)$). $(\mathrm{IO}_w\, A)$ is a near variant of the "W-type" in standard type theory: The type expression $\mathrm{W}x : A.B$ denotes the type of well-founded trees with nodes labelled by elements $a : A$ and having then branching degree $B[x := a]$, see [9, pages 109–114]and [7, pages 79–86] for details. In proof theory, the W-type turns out to be a very powerful construction: see [11]

Execution of I/O-programs. Up to now we have defined an inductive data type of I/O-programs within constructive type theory, but there is still no way to actually *run* such a program. Execution is an *external* operation rather than a constant within type theory. Just as an implementation of type theory will provide an external operation or facility to compute (and display) the (head-) normal form of a term, so we propose to provide a second operation that executes a term denoting an I/O-program.

More precisely, this works as follows. Let $w_0 = \langle C_0, R_0 \rangle$ be a world corresponding to the real commands, so that to every $c : C_0$ there corresponds a real I/O-command having some value $r : R_0\, c$ as result. If we have derived $p : \mathrm{IO}_{w_0}\, A$ then the external operation $\mathsf{execute}$ can be performed upon p. The operation $\mathsf{execute}$ does the following. It reduces p to canonical form, i.e. to a term of constructor form. This form must be either $(\mathsf{leaf}\, a)$ or $(\mathsf{do}\, c\, q)$. If it is $(\mathsf{leaf}\, a)$, then $a : A$ and execution terminates, yielding as result a (which, when running the program from a command line will be displayed in a similar way as the result of the evaluation of an expression). If it is $(\mathsf{do}\, c\, q)$, then first the interactive command corresponding to c is performed obtaining a result $r : R\, c$, after which execution continues with $(q\, r)$.

Roughly speaking a program p is evaluated to normal form, as it were 'fetching' the next instruction. The instruction is 'executed', and the result used to select the next program to be evaluated. So through successive interactions we trace out a descending chain through the tree p.

A first example. In the following example we assume commands $\mathsf{readstr}$ and $(\mathsf{writestr}\, s)$ for reading and writing strings and a Boolean valued equality $=_{\mathsf{String}}$ on strings. The following program prompts for the root-password. If the user types in the right one ("Wurzel"[3]) the program terminates successfully, otherwise

[3] This really happened.

it responds with "Login incorrect" and fails. We use some syntactic sugar.

$$
\begin{aligned}
C &= \{\,\mathsf{readstr}\,\} \cup \{\,\mathsf{writestr}\,s \mid s : \mathsf{String}\,\} : \mathrm{Set}\ , \\
R &: C \to \mathrm{Set}\ , \\
R\,\mathsf{readstr} &= \mathsf{String}\ , \\
R\,(\mathsf{writestr}\,s) &= \mathbf{1}\ , \\
\mathrm{Wurzel} &= \mathsf{do}\ \mathsf{writestr}\ \text{“Password (root):”} \\
&\qquad \lambda a.\,\mathsf{do}\ \mathsf{readstr} \\
&\qquad\quad \lambda s.\,\mathbf{if}\ s =_{\mathsf{String}} \text{“Wurzel”} \\
&\qquad\qquad \mathbf{then}\ \mathsf{leaf}\,\mathrm{success} \\
&\qquad\qquad \mathbf{else}\ \mathsf{do}\ (\mathsf{writestr}\ \text{“Login incorrect”}) \\
&\qquad\qquad\qquad \lambda a.\,\mathsf{leaf}\,\mathrm{fail} \\
&: \mathrm{IO}\,\{\,\mathrm{success}, \mathrm{fail}\,\}\ .
\end{aligned}
$$

η, $*$. It is now easy to define η and $*$ and verify the monad laws for well-founded trees with extensional equality (by [4] this is not the most efficient solution):

$$
\begin{aligned}
\eta^A_a &= \mathsf{leaf}\,a\ , \\
\mathsf{leaf}\,a *_{A,B} q &= q\,a\ , \\
\mathsf{do}\,c\,p *_{A,B} q &= \mathsf{do}\,c\,(\lambda x.p\,x *_{A,B} q)\ .
\end{aligned}
$$

3 Constructions for Defining I/O-Trees

It should be possible to define interactive programs with infinitely many interactions. For instance, if we execute an editor and never terminate the program, the execution should go on forever. So we need constructions for defining such programs. This will however destroy normalisation. We will see in Sect. 4 how to modify the concept in order to obtain a normalising type theory. The definitions of all constructions in this section are possible only in the presence of *dependent types*, which demonstrate their expressive power.

repeat. Assume $A, B : \mathrm{Set}$, $b : B$, $p : B \to \mathrm{IO}_w\,(B + A)$. We want to define a program $\mathsf{repeat}_A\,B\,b\,p : \mathrm{IO}_w\,A$, which, when executed, operates as follows. First, program $(p\,b)$ is executed. If it has result $(\mathrm{inl}\,b')$, then the program continues with $(\mathsf{repeat}_A\,B\,b'\,p)$. If it has result $(\mathrm{inr}\,a)$, the program terminates and returns a.

However, if $p\,b = \mathsf{leaf}\,a$ for some $b : B$, we might get an expression that does not evaluate to constructor form, e.g. $(\mathsf{repeat}_B\,b\,\lambda x.\mathsf{leaf}\,(\mathrm{inl}\,b))$. So we have to restrict p, and the easiest way is to replace $(\mathrm{IO}_w\,(B + A))$ by $(\mathrm{IO}^+_w\,(B + A))$. Here for $D : \mathrm{Set}$ let $\mathrm{IO}^+_w\,D = \Sigma c : C.R\,c \to \mathrm{IO}_w\,D$ be the set of I/O-programs with results in D, with at least one interaction (command c). Let $\mathsf{do}^+\,c\,p = \langle c, p\rangle : \mathrm{IO}^+_w\,D$ and, if $p : \mathrm{IO}^+_w\,D$, let $p^- : \mathrm{IO}_w\,D$ be defined by $(\mathsf{do}^+\,c\,p)^- = \mathsf{do}\,c\,p$.

The definition (which uses general recursion and so allows us to define non-well-founded trees and form non-normalising terms) of repeat is as follows:

$$
\mathsf{repeat}_w\ :\ (A, B : \mathrm{Set}, b : B, p : B \to \mathrm{IO}^+_w\,(B + A)) \to \mathrm{IO}_w\,B\ ,
$$

$$\begin{aligned}\mathsf{repeat}_{w,A}\, B\, b\, p &= (p\, b)^{-} *_{w,B+A,A}\, q \ , \\ \textbf{where} \quad q\,(\mathrm{inl}\, b') &= \mathsf{repeat}_{w,A}\, B\, b'\, p \ , \\ q\,(\mathrm{inr}\, a) &= \mathsf{leaf}\, a \ .\end{aligned}$$

Example. As an example we define a rudimentary editor. The only command is readChar, which has as result either a character c typed in, cursorLeft for the cursor-left-button or done for some key associated with termination. The program reads the text created using these keys and returns the result. ((truncate s) will be the result of deleting the last character from string s, (append $s\, c$) an operation which appends character c to the end of string s, and "" the empty string.)

$$\begin{aligned} C &= \{\mathsf{readChar}\} : \mathrm{Set} \ , \\ R &: C \to \mathrm{Set} \ , \\ R\, c &= \{\mathsf{ch}\, c \mid c : \mathsf{Char}\} \cup \{\mathsf{cursorLeft}, \mathsf{done}\} \ . \\ \mathsf{editor} &= \mathsf{repeat}_{\langle C,R\rangle,\mathsf{String}}\, \mathsf{String}\ \text{“”}\ \lambda s.\mathsf{do}^{+}\, \mathsf{readChar}\, q \ , \\ \textbf{where} \quad q\,(\mathsf{ch}\, c) &= \mathsf{leaf}\,(\mathrm{inl}\,(\mathsf{append}\, s\, c)) \ , \\ q\, \mathsf{cursorLeft} &= \mathsf{leaf}\,(\mathrm{inl}\,(\mathsf{truncate}\, s)) \ , \\ q\, \mathsf{done} &= \mathsf{leaf}\,(\mathrm{inr}\, s) \ . \end{aligned}$$

While loop. While loops are defined similarly to repeat loops.

$$\mathsf{while}_w : (A, B : \mathrm{Set}, b : B, p : B \to (\mathrm{IO}^{+}_w\, B + \mathrm{IO}_w\, A)) \to \mathrm{IO}_w\, A \ .$$

The definition proceeds by cases on the value of $(p\, b)$. If it is of the form $(\mathrm{inl}\, q)$, then q is executed, and, once it terminates with result b', the program continues with $(\mathsf{while}_{w,A}\, B\, b'\, p)$. If it is $(\mathrm{inr}\, q)$, q is executed and its result returned as final result. The definition, which uses again general recursion, is

$$\begin{aligned}\mathsf{while}_{w,A}\, B\, b\, p &= f\,(p\, b) \ , \\ \textbf{where} \quad f\,(\mathrm{inl}\, q) &= q^{-} *_{w,B,A}\, \lambda b'.\mathsf{while}_{w,A}\, B\, b'\, p \ , \\ f\,(\mathrm{inr}\, q) &= q \ .\end{aligned}$$

It is now an easy exercise to express while by repeat and vice versa.

Redirect. $*$ can be regarded as "horizontal composition" of programs. There is also a "vertical composition": Assume worlds $w = \langle C, R\rangle$ and $w' = \langle C', R'\rangle$, $A : \mathrm{Set}$ and $p : \mathrm{IO}_w\, A$. We want to refine p to a program in world w', by replacing every command $c : C$ by a program $(q\, c)$ in world w' with a result $r : R\, c$. So q has type $(c : C) \to \mathrm{IO}_{w'}(R\, c)$. However, if we allow $(q\, c)$ to be a leaf and p has infinitely many commands, this will allow us to construct an expression that cannot be evaluated to constructor form. To avoid this, we replace the type of q by $(c : C) \to \mathrm{IO}^{+}_{w'}(R\, c)$. The construction that results is

$$\begin{aligned}&\mathsf{redirect}_{w,w'} : (A : \mathrm{Set}, p : \mathrm{IO}_w\, A, q : (c : C) \to \mathrm{IO}^{+}_{w'}(R\, c)) \to \mathrm{IO}_{w'}\, A \ , \\ &\quad \textbf{where} \quad \mathsf{redirect}_{w,w',A}\,(\mathsf{leaf}\, a)\, q = \mathsf{leaf}\, a \ ,\end{aligned}$$

$$\mathsf{redirect}_{w,w',A}\,(\mathsf{do}\,c\,p)\,q = (q\,c)^{-} *_{w',R\,c,A} \lambda r.\mathsf{redirect}_{w,w',A}\,(p\,r)\,q \ .$$

Using redirect *for building libraries.* We can now define a world in which high level I/O-commands are first class objects – they do not evaluate directly into low level commands – together with an interpretation of each command as a program in the basic language used by execute, and so construct libraries. To implement execute one can therefore restrict oneself to a basic world with simple commands.

Example. Let the high level world be $w_0 = \langle C_0, R_0\rangle$, with $C_0 = \{\mathsf{read}\} \cup \{\mathsf{write}\,s \mid s : \mathsf{String}\}$. Here read is a command for reading a string, $R_0\,\mathsf{read} = \mathsf{String}$, and $(\mathsf{write}\,s)$ an instruction for writing a string, $R_0\,(\mathsf{write}\,s) = \mathbf{1}$. Let the low level world w_1 have commands for reading a key, writing a symbol, and movements of the cursor left and right. Let $q : (c : C_0) \to \mathrm{IO}_{w_1}\,(R_0\,c)$, where $(q\,\mathsf{read})$ is an editor that uses the keys to manipulate a string and has as result that string, and $(q\,(\mathsf{write}\,s))$ is an output routine for strings. Then $(\mathsf{redirect}\,p\,q)$ translates a program using high level commands into one that uses the basic ones.

Equality. With while− and repeat−loops we introduce non-well-founded I/O-trees. Even with extensional equality it seems that it is no longer possible to prove the monad laws. (We do not yet have a proof of this.) So extensional equality seems to be too weak for dealing with non-well-founded programs. Instead we use bisimulation as equality. In [5] I. Lindström has given a very elegant definition of such an equality. The definition is based on an idea that occurs in work on non-wellfounded sets by Lars Hallnäs [2]. Transferred to our setting, the equality is defined as $\forall n.p \simeq'_{w,A,n} q$, where $p \simeq'_{w,A,n} q$ expresses that p and q coincide up to height n. In the following the world w will be a parameter in all definitions, and will be omitted for clarity. We will use equality-types $=_C$ and $=_A$ on C and A. (We will in a follow-up to this article consider a generalisation where instead of assuming $=_C$ we establish C with a setoid structure; in this case we need a reindexing map, which replaces $\mathrm{J}_C R$ below. Additional reindexing maps will be needed to establish the properties of the equality which we define.)

$$\begin{aligned}
\simeq &: (A : \mathrm{Set}, p, q : \mathrm{IO}\,A) \to \mathrm{Set} \ ,\\
\simeq' &: (A : \mathrm{Set}, n : \mathbb{N}, p, q : \mathrm{IO}\,A) \to \mathrm{Set} \ ,\\
(p \simeq_A q) &= \forall n : \mathbb{N}.p \simeq'_{A,n} q,\\
(p \simeq'_{A,0} q) &= \top \ ,\\
(\mathsf{leaf}\,a \simeq'_{A,n+1} \mathsf{do}\,c\,p) &= (\mathsf{do}\,c\,p \simeq'_{A,n+1} \mathsf{leaf}\,a) = \bot \ ,\\
(\mathsf{leaf}\,a \simeq'_{A,n+1} \mathsf{leaf}\,a') &= (a =_A a') \ ,\\
(\mathsf{do}\,c\,p \simeq'_{A,n+1} \mathsf{do}\,c'\,p') &= \exists x : (c =_C c').\forall r : R\,c.p\,r \simeq'_{A,n} p'\,(\mathrm{J}_C\,R\,c\,c'\,x\,r) \ .
\end{aligned}$$

Definition 3.1. *(a) Let case-distinction for* IO *be the rule (under the assumptions that* $A : \mathrm{Set}$*,* $B : (p : \mathrm{IO}\,A) \to \mathrm{Set}$*):*

$$\begin{aligned}
\mathrm{C}^{\mathrm{IO}}_{A,B} : ((a : A) \to B\,(\mathsf{leaf}\,a), (c : C, q : R\,c \to \mathrm{IO}\,A) \to B\,(\mathsf{do}\,c\,q),\\
p : \mathrm{IO}\,A) \to B\,p \ .
\end{aligned}$$

(b) Let TT(IO) *be (intensional) Martin-Löf type theory extended by the defining rules for* IO *and case-distinction for* IO.

Lemma 3.2. TT(IO) proves the following (under the assumptions that A, B : Set and all other variables are of appropriate type)

(a) $\simeq_A$ is reflexive, symmetric and transitive.
(b) $p \simeq_A p' \to (\forall a : A.q\,a \simeq_B q'\,a) \to p *_{A,B} q \simeq_A p' *_{A,B} q'$.

Proof. (a): First we prove the lemma with $\simeq_A$ replaced by $\simeq'_{A,n}$ by induction on $n : \mathbb{N}$, using the elimination rules for equality. Then the assertion follows by the definition of $\simeq$. (b) Show that $p \simeq'_{A,n} p'$ and $\forall a : A.q\,a \simeq'_{B,m} q'\,a$ imply $\mathsf{do}\,p\,q \simeq'_{B,\min\{n,m\}} \mathsf{do}\,p'\,q'$ by induction on n. □

Theorem 3.3. TT(IO) proves the monad laws with respect to $\simeq_A$.

Proof. The first law holds definitionally and by reflexivity therefore with respect to $\simeq_A$. The second and third laws are proved first with $\simeq_A$ replaced by $\simeq'_{A,n}$ by induction on n. Then the assertion follows from the definition of $\simeq_A$. □

I/O-trees as a general concept for command/response-interaction. It seems that the applications of I/O-trees, which are in general non-well-founded trees, are not limited only to functional programming languages. I/O-trees cover in a general way command/response-interaction with one agent (a program) having control over the commands. Every I/O-behaviour corresponds, up to the equality we have introduced above, to exactly one I/O-tree. Therefore I/O-trees are suitable models for this kind of interaction.

4 Normalising Version

Counterexample to normalisation. If we take standard reduction rules corresponding to the equalities given above (by directing the equations in an obvious way), the above definitions give non-normalising programs. Let for instance $A = B = C = \mathbb{N}$, $(R\,c)$ be arbitrary, $w = \langle C, R\rangle$, $f : \mathbb{N} \to \mathbb{N}$. We omit the parameter w.

$$p := \lambda n.\mathsf{do}^+\,(f\,n)\,\lambda x.\mathsf{leaf}\,(\mathrm{inl}\,(n+1)) : \mathbb{N} \to \mathrm{IO}^+\,(A+B)\ ,$$

$$\begin{aligned}
\mathsf{repeat}\,0\,p &\longrightarrow \mathsf{do}\,(f\,0)\,\lambda x.\mathsf{repeat}\,(\mathrm{S}\,0)\,p \\
&\longrightarrow \mathsf{do}\,(f\,0)\,\lambda x.\mathsf{do}\,(f\,(\mathrm{S}\,0))\,\lambda y.\mathsf{repeat}\,(\mathrm{S}\,(\mathrm{S}\,0))\,p \\
&\longrightarrow \mathsf{do}\,(f\,0)\,\lambda x.\mathsf{do}\,(f\,(\mathrm{S}\,0))\,\lambda y.\mathsf{do}\,(f\,(\mathrm{S}\,(\mathrm{S}\,0)))\,\lambda z.\mathsf{repeat}\,(\mathrm{S}\,(\mathrm{S}\,(\mathrm{S}\,0)))\,p \\
&\longrightarrow \cdots\ .
\end{aligned}$$

We see that definitional equality is now undecidable, since we cannot decide whether two functions $\mathbb{N} \to \mathbb{N}$ are extensionally equal. This implies the undecidability of type checking, since with an type checking algorithm we can

decide definitional equality (for $a, b : A$, the term $\lambda B, f.f\,a$ is of type $(B : (x : A) \to \mathrm{Set}, f : (x : A) \to B\,x) \to B\,b$ if and only if $a = b : A$).

One solution would be to extend dependent type theory by coinductive types with rules chosen such that normalisation is preserved. This requires extensive meta-theoretical investigations that have not yet been completely carried out. Instead we represent non-well-founded trees in normalising standard type theory.

How to regain normalisation. In type theory with inductive types and standard elimination rules for them, while and repeat cannot be defined. We can however add one of them as a constructor to $(\mathrm{IO}_w\,A)$. We choose while, for which the definition of $*$ and the proofs of equalities turn out to be easier. We can then define repeat by using while. We modify execute, so that it operates on $(\mathsf{while}\,u\,a\,p)$ in the same way as it operated on the non-well-founded trees defined using the **function** while in the previous version. One problem is however that while (the same is the case with repeat) defines an element of $(\mathrm{IO}_w\,A)$ by referring to $(\mathrm{IO}_w\,B)$ for an arbitrary set B. To demand that $(\mathrm{IO}_w\,A)$ is a set means to define a set by referring negatively to all sets, which is problematic. (The typing rules require that if A is a set, $(\mathrm{IO}_w\,A)$ is a type).

To fix this, we will restrict the sets referred to in while to elements of a universe. A universe is a set-indexed collection of sets, ie. a pair $\langle U, T\rangle$ s.t. $U : \mathrm{Set}$ and $T : U \to \mathrm{Set}$. The elements of U represent "small sets". With such a restriction $(\mathrm{IO}_w\,A)$ no longer refers to the collection of all sets, and can now be typed as a set. We will however extend U to a slightly bigger universe with representatives for $\mathbf{1} + R\,c$, and this extension will be called set, since it is in the definition of $(\mathrm{IO}_w\,A)$ the "collection of small sets".

General assumption and definition 4.1. *(a) Let $w = \langle C, R\rangle$ be a world.*
(b) Let $\mathrm{U} : \mathrm{Set}$, $\mathrm{T} : \mathrm{U} \to \mathrm{Set}$ *be some fixed collection of sets (i.e. a universe).*
(c) Let $\mathrm{set} := \mathrm{U} + C$, $\mathrm{el} : \mathrm{set} \to \mathrm{Set}$, $\mathrm{el}\,(\mathrm{inl}\,u) = \mathrm{T}\,u$, $\mathrm{el}\,(\mathrm{inr}\,c) = \mathbf{1} + R\,c$, *$R$ according to the world w. We write* $\widehat{(\mathbf{1} + \mathrm{R}\,c)}$ *for* $(\mathrm{inr}\,c)$.

For simplicity, in the following we will omit the parameters w, U, and T.

We can now omit the constructor do (which can be simulated by while) and obtain the following definition of $\mathrm{IO}\,A$:

$$
\begin{aligned}
\mathrm{IO} &: \mathrm{Set} \to \mathrm{Set}\ ,\ \textbf{where}\ (\mathrm{IO}\,A)\ \textbf{has constructors}\\
\mathsf{leaf} &: A \to \mathrm{IO}\,A\ ,\\
\mathsf{while} &: (u : \mathrm{set}, a : \mathrm{el}\,u, n : \mathrm{el}\,u \to (\mathrm{IO}^+\,(\mathrm{el}\,u) + \mathrm{IO}\,A)) \to \mathrm{IO}\,A\ ,\\
\textbf{and}\ \mathrm{IO}^+ &: \mathrm{Set} \to \mathrm{Set}\ ,\\
\mathrm{IO}^+\,A &= \Sigma c : C.R\,c \to \mathrm{IO}\,A\ .
\end{aligned}
$$

Monad operations. In the monad operations sets have to be replaced by elements of the universe:

$$
\begin{aligned}
\eta^A_a &:= \mathsf{leaf}\,a\ ,\\
\mathsf{leaf}\,a *_{A,B} q &= q\,a\ ,
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{while}\, u\, a\, p *_{A,B} q \;=\; \mathsf{while}\, u\, a\, (p \circledast_{A,B,u} q) \;\;, \\
&\quad \textbf{where } \circledast \;:\; (A, B : \mathrm{Set}, u : \mathrm{set}, p : \mathrm{el}\, u \to (\mathrm{IO}^+ (\mathrm{el}\, u) + \mathrm{IO}\, A), \\
&\qquad\qquad q : A \to \mathrm{IO}\, B) \to \mathrm{el}\, u \to (\mathrm{IO}^+ (\mathrm{el}\, u) + \mathrm{IO}\, B) \;\;, \\
&\qquad\qquad \text{if } p\, b = \mathrm{inl}\, p', \text{ then } (p \circledast_{A,B,u} q)\, b = \mathrm{inl}\, p' \;\;, \\
&\qquad\qquad \text{if } p\, b = \mathrm{inr}\, p', \text{ then } (p \circledast_{A,B,u} q)\, b = \mathrm{inr}\, (p' *_{A,B} q) \;\;.
\end{aligned}
$$

Do. Now we define the operation $(\mathsf{do}_A\, c\, p)$. (Note that do is *not* a constructor):

$$
\begin{aligned}
&\mathsf{do}_A\, c\, p = \mathsf{while}\, (\widehat{\mathbf{1} + \mathrm{R}\, c})\, (\mathrm{inl}\, \bullet)\, q \;\;, \\
&\textbf{where} \quad q\, (\mathrm{inl}\, \bullet) = \mathrm{inl}\, \langle c, \lambda r.\mathsf{leaf}\, (\mathrm{inr}\, r) \rangle \;\;, \\
&\qquad\quad\; q\, (\mathrm{inr}\, r) = \mathrm{inr}\, (p\, r) \;\;.
\end{aligned}
$$

Split. In the non-normalising theory, each element of $(\mathrm{IO}\, A)$ according to the new definition can be interpreted as a non-well-founded tree; we replace all occurrences of the constructor while with the function while defined before. In normalising type theory this is not possible. Instead we can obtain the structure of the represented non-well-founded trees by defining a function split_A, which determines for every $p : \mathrm{IO}\, A$ whether its interpretation as a non-well-founded tree is that of a leaf labelled by $a : A$ ($\mathrm{split}_A\, p = \mathrm{inr}\, a$) or whether it is an inner node labelled by $c : C$, which has for $r : R\, c$ subtree $q\, r$ ($\mathrm{split}_A\, p = \mathrm{inl}\, \langle c, q \rangle$):

$$
\begin{aligned}
&\mathrm{split} \;:\; (A : \mathrm{Set}, p : \mathrm{IO}\, A) \to (\mathrm{IO}^+\, A + A) \;\;, \\
&\mathrm{split}_A\, (\mathsf{leaf}\, a) = \mathrm{inr}\, a \;\;, \\
&\textbf{If } \; p\, a = \mathrm{inr}\, q, \qquad \textbf{then } \mathrm{split}_A\, (\mathsf{while}\, u\, a\, p) = \mathrm{split}_A\, q \;\;, \\
&\textbf{If } \; p\, a = \mathrm{inl}\, \langle c, q \rangle, \textbf{ then} \\
&\mathrm{split}_A\, (\mathsf{while}\, u\, a\, p) = \mathrm{inl}\, \langle c, \lambda r.q\, r *_{\mathrm{el}(u),A} \lambda x.\mathsf{while}\, u\, x\, p \rangle \;\;.
\end{aligned}
$$

Execution of I/O-programs. Assume a fixed world $w_0 = \langle C_0, R_0 \rangle$ corresponding to real commands, as before. $\mathsf{execute}$, adapted to the new setting, operates as follows: Applied to a program $p : \mathrm{IO}_{C_0,R_0}\, A$ it evaluates $\mathrm{split}_A\, p$. If the result is $(\mathrm{inr}\, a)$, then $\mathsf{execute}$ stops with result a. If $\mathrm{split}_A\, p = \mathrm{inl}\, \langle c, q \rangle$, then c is executed, and depending on the result r, $\mathsf{execute}$ continues with $(q\, r)$.

Normalising I/O-programs. With only inductive data types with their elimination rules, type theory is normalising. Therefore $\mathrm{split}_A\, q$ reduces to a value of the form $(\mathrm{inr}\, a)$ or $(\mathrm{inl}\, \langle c, q \rangle)$. So when a program is executed, and it is its 'turn to go' (ie. at the beginning and after obtaining a response to a command), after a finite time, either it terminates, or it issues another command. (Whether a response to a command c is obtained after a finite time depends firstly on whether a response is even possible – the response set $R\, c$ may be empty – and secondly on what happens in the real world – the user may walk away from the keyboard and never return.) However, it may still be that infinitely many commands are executed. As trees, I/O-programs are not necessarily well-founded. We call an I/O-program *normalising* if both initially and after the result of a command is obtained, it either terminates, or issues the next command after a finite amount

of time. The set (IO A) (together with execute) represents a class of normalising I/O-programs.

Equality. Under the same assumptions as in Sect. 3 we can define now an equality on elements of IO A. However, we use split in order to get access to the corresponding tree-structure:

$$
\begin{aligned}
\simeq &: (A : \mathrm{Set}, p, q : \mathrm{IO}\, A) \to \mathrm{Set}\ , \\
\simeq' &: (A : \mathrm{Set}, n : \mathbb{N}, p, q : \mathrm{IO}\, A) \to \mathrm{Set}\ , \\
(p \simeq_A q) &= \forall n : \mathbb{N}.p \simeq'_{A,n} q\ , \\
p \simeq'_{A,0} q &= \top\ , \\
p \simeq'_{A,n+1} q &= (\mathrm{split}_A\, p \simeq''_{A,n} \mathrm{split}_A\, q)\ , \ \textbf{where} \\
\simeq'' &: (A : \mathrm{Set}, n : \mathbb{N}, p, q : \mathrm{IO}^+\, A + A) \to \mathrm{Set}\ , \\
(\mathrm{inr}\, a \simeq''_{A,n} \mathrm{inl}\, \langle c, p\rangle) &= (\mathrm{inl}\, \langle c, p\rangle \simeq''_{A,n} \mathrm{inr}\, a) = \bot\ , \\
(\mathrm{inr}\, a \simeq''_{A,n} \mathrm{inr}\, a') &= (a =_A a')\ , \\
(\mathrm{inl}\, \langle c, q\rangle \simeq''_{A,n} \mathrm{inl}\, \langle c', q'\rangle) &= \exists p : (c =_C c').\forall r : R\, c.q\, r \simeq'_{A,n} q'\, (\mathrm{J}\, C\, R\, c\, c'\, p\, r)\ .
\end{aligned}
$$

Note that $\simeq'_{A,n}$ identifies programs which behave identically in the first n steps, and therefore $\simeq_A$ identifies exactly behaviourally equal programs. Note however that we identify only those commands $c : C$ which are equal with respect to $=_C$.

Proof of the monad laws, defining equalities for while *and other standard properties with respect to bisimulation.* The following can be proved inside type theory. (Some indices or superscripts have been left implicit).

Lemma 4.2. (a) $\eta_a * p \simeq_A p\, a$.
(b) $\simeq_A$ and $\simeq'_{A,n}$ are reflexive, symmetric and transitive.
(c) If $p\, a =_{\mathrm{IO}\,(\mathrm{el}\, u)+\mathrm{IO}\, A} \mathrm{inr}\, q$, then $\mathsf{while}\, u\, a\, p \simeq_A q$.

Proof. (a) is trivial. (b) follows with $\simeq_A$ replaced by $\simeq'_{A,n}$ by induction on n – in case of symmetry and transitivity one uses additionally the elimination rules for $=_C$. From this the assertion follows. (c) $\mathrm{split}\, (\mathsf{while}\, u\, a\, p) =_{\mathrm{IO}^+\, A+A} \mathrm{split}\, q$. □

For stating and proving the next lemmata we introduce an equality on the type of p in $(\mathsf{while}\, u\, a\, p)$, i.e. $(\mathrm{el}\, u) \to (\mathrm{IO}^+\, (\mathrm{el}\, u) + A)$:

Definition 4.3.

$$
\begin{aligned}
\simeq^{\mathrm{w}} &: (A : \mathrm{Set}, u : \mathrm{set}, \\
&\qquad p, q : (\mathrm{el}\, u) \to (\mathrm{IO}^+\, (\mathrm{el}\, u) + \mathrm{IO}\, A)) \to \mathrm{Set}\ , \\
p \simeq^{\mathrm{w}}_{A,u} q &= \forall x : \mathrm{el}\, u.p\, x \simeq^{\mathrm{w,aux}}_{A,u} q\, x\ , \ \textbf{where} \\
\simeq^{\mathrm{w,aux}} &: (A : \mathrm{Set}, u : \mathrm{set}, p, q : \mathrm{IO}^+\, (\mathrm{el}\, u) + \mathrm{IO}\, A) \\
&\qquad \to \mathrm{Set}\ , \\
(\mathrm{inl}\, q \simeq^{\mathrm{w,aux}}_{A,u} \mathrm{inr}\, q) &= (\mathrm{inr}\, q \simeq^{\mathrm{w,aux}}_{A,u} \mathrm{inl}\, q) = \bot\ , \\
(\mathrm{inr}\, q \simeq^{\mathrm{w,aux}}_{A,u} \mathrm{inr}\, q) &= (q \simeq_A q')\ , \\
(\mathrm{inl}\, \langle c, q\rangle \simeq^{\mathrm{w,aux}}_{A,u} \mathrm{inl}\, \langle c', q'\rangle) &= \exists p : (c =_C c').\forall r : R\, c. \\
&\qquad q\, r \simeq_{\mathrm{el}\, u} q'\, (\mathrm{J}\, C\, R\, c\, c'\, p\, r)\ .
\end{aligned}
$$

Similarly we define $\simeq^{\mathrm{w}\prime}$, $\simeq^{\mathrm{w,aux}\prime}$ *with an additional argument* $n : N$ *and refer to* $\simeq'_{A,n}$, $\simeq'_{\mathrm{el}\,u,n}$ *instead of* $\simeq_A$, $\simeq_{\mathrm{el}\,u}$.

Lemma 4.4. (a) $(p_0 \simeq_A p_1 \wedge \forall a : A.q_0\,a \simeq_B q_1\,a) \to (p_0 * q_0 \simeq_B p_1 * q_1)$.
(b) $p_0 \simeq^{\mathrm{w}}_{A,u} p_1 \to \mathsf{while}\,u\,a\,p_0 \simeq_A \mathsf{while}\,u\,a\,p_1$.
(c) For $p : \mathrm{IO}\,A$, $q : A \to \mathrm{IO}\,B$, $r : B \to \mathrm{IO}\,D$ it follows
$(p * q) * r \simeq_D p * \lambda x.((q\,x) * r)$.

Proof. We prove (a) – (c), with $\lambda x.\simeq_x$, $\lambda x.\simeq^{\mathrm{w}}_{x,u}$ replaced by $\lambda x.\simeq'_{x,n}$, $\lambda x.\simeq'^{\mathrm{w}}_{x,u,n}$, simultaneously by induction on n. The case $n = 0$ is trivial, so we assume the assertion has been proved for n and prove it for $n+1$:
(a) Side-induction on p_0, side-side-induction on p_1:
If $p_0 = \mathsf{leaf}\,a_0$ and $p_1 = \mathsf{leaf}\,a_1$, then $a_0 =_A a_1$ and $q_0\,a_0 \simeq'_{B,n+1} q_1\,a_1$.
If $p_0 = \mathsf{while}\,u\,a\,\widetilde{p}_0$ with $\widetilde{p}_0\,a = \mathrm{inr}\,\widehat{p}_0$, then $p_0 \simeq_A \widehat{p}_0$, and by
$p_0 * q_0 = \mathsf{while}\,u\,a\,(\widetilde{p}_0 \circledast q_0)$, $(\widetilde{p}_0 \circledast q_0)\,a = \mathrm{inr}\,(\widehat{p}_0 * q_0)$ it follows
$p_0 * q_0 \simeq_B \widehat{p}_0 * q_0$, and the assertion follows by side-IH for $\widehat{p}_0$ instead of p_0.
Similarly the assertion follows if p_1 is of a similar form.
Otherwise $p_i = \mathsf{while}\,u_i\,a_i\,\widetilde{p}_i$, with $\widetilde{p}_i\,a_i = \mathrm{inl}\,\langle c_i, \widehat{p}_i\rangle$,
$\mathrm{split}\,p_i = \mathrm{inl}\,\langle c_i, \lambda r.\widehat{p}_i\,r * \lambda x.\mathsf{while}\,u_i\,x\,\widetilde{p}_i\rangle$.
By $p_i \simeq_{A,n+1} p'_i$ there exists $p_{c_0c_1} : (c_0 =_C c_1)$, and for
$r_0 : R\,c_0$, $r_1 := \mathrm{J}\,C\,R\,c_0\,c_1\,p_{c_0c_1}\,r_0$ there exist proofs of
$\widehat{p}_0\,r_0 * \lambda x.\mathsf{while}\,u_0\,x\,\widetilde{p}_0 \simeq'_{A,n} \widehat{p}_1\,r_1 * \lambda x.\mathsf{while}\,u_1\,x\,\widetilde{p}_1$.
$\mathrm{split}\,(p_i * q_i) = \mathrm{inl}\,\langle c_i, \lambda r.\widehat{p}_i\,r * \lambda x.\mathsf{while}\,u_i\,x\,(\widetilde{p}_i \circledast q_i)\rangle$
$= \mathrm{inl}\,\langle c_i, \lambda r.\widehat{p}_i\,r * \lambda x.\mathsf{while}\,u_i\,x\,\widetilde{p}_i * q_i\rangle$.
We have to show that for r_0, r_1 as above
$\widehat{p}_0\,r_0 * \lambda x.\mathsf{while}\,u_0\,x\,\widetilde{p}_0 * q_0 \simeq'_n \widehat{p}_1\,r_1 * \lambda x.\mathsf{while}\,u_1\,x\,\widetilde{p}_1 * q_1$.
By IH (c) and symmetry $\widehat{p}_i\,r_i * \lambda x.\mathsf{while}\,u_i\,x\,\widetilde{p}_i * q_i \simeq'_{B,n} (\widehat{p}_i\,r_i * \lambda x.\mathsf{while}\,u_i\,x\,\widetilde{p}_i) * q_i$,
and by IH (a) $(\widehat{p}_0\,r_0 * \lambda x.\mathsf{while}\,u_0\,x\,\widetilde{p}_0) * q_0 \simeq'_{B,n} (\widehat{p}_1\,r_1 * \lambda x.\mathsf{while}\,u_1\,x\,\widetilde{p}_1) * q_1$.
The assertion follows now by transitivity and symmetry.
(b) If $p_i\,a_i = \mathrm{inr}\,q_i$, then $\mathsf{while}\,u\,a_i\,p_i = q_i$, $q_0 \simeq'_{A,n+1} q_1$.
Otherwise $p_i\,a_i = \mathrm{inl}\,\langle c_i, q_i\rangle$, $\mathrm{split}\,(\mathsf{while}\,u\,a_i\,p_i) = \mathrm{inl}\,\langle c_i, \lambda r.q_i\,r * \lambda x.\mathsf{while}\,u\,x\,p_i\rangle$.
By assumption there exists p_{c_0,c_1} as in (a) and for r_0 and r_1 as in (a) proofs of $q_0\,r_0 \simeq'_{\mathrm{el}\,u} q_1\,r_1$, and furthermore by IH (b) proofs of $\mathsf{while}\,u\,x\,p_0 \simeq'_{A,n}$ $\mathsf{while}\,u\,x\,p_1$ for $x : \mathrm{el}(u)$. The assertion follows by IH (a).
(c) is proved by side-induction on p. If $p = \mathsf{leaf}\,a$ this follows by reflexivity.
Otherwise $p = \mathsf{while}\,u\,a\,\widetilde{p}$, $(p * q) * r = \mathsf{while}\,u\,a\,((\widetilde{p} \circledast q) \circledast r)$, $p * \lambda x.q\,x * r =$ $\mathsf{while}\,u\,a\,(\widetilde{p} \circledast \lambda x.q\,x \circledast r)$, by side-IH $(\widetilde{p} \circledast q) \circledast r \simeq^{\mathrm{w}\prime}_{D,u,n+1} \widetilde{p} \circledast \lambda x.q\,x \circledast r$,
and by (b) for $n+1$, as just proved, the assertion follows. □

Lemma 4.5. (a) $p * \lambda x.\eta_x \simeq_A p$.
(b) $\mathrm{split}\,(\mathsf{do}\,c\,p) =_{\mathrm{IO}^+\,A+A} \mathrm{inl}\,\langle c, p'\rangle$ for some p' s.t. $\forall r : R\,c.p\,r \simeq_A p'\,r$.
(c) If $p\,a =_{\mathrm{IO}^+\,(\mathrm{el}\,u)+\mathrm{IO}\,A} \mathrm{inl}\,\langle c, q\rangle$ then
$\mathsf{while}\,u\,a\,p \simeq_A \mathsf{do}\,c\,\lambda r.q\,r * \lambda x.\mathsf{while}\,u\,x\,p$.

Proof. (a) follows by straightforward induction on p and Lemma 4.4 (b).
(b) $\mathrm{split}\,(\mathsf{do}\,c\,p) = \langle c, \lambda r.(\mathsf{leaf}\,(\mathrm{inr}\,r)) * \lambda x.\mathsf{while}\,(\widehat{\mathbf{1} + \mathrm{R}\,c})\,x\,q\rangle$

$= \langle c, \lambda r.\mathsf{while}\,(\widehat{\mathbf{1} + \mathrm{R}\,c})\,(\mathrm{inr}\,r)\,q\rangle,$
where q is as in the definition of $(\mathsf{do}\,c\,p)$. For $r : R\,c$ we have by Lemma 4.2 (c) $\mathsf{while}\,(\widehat{\mathbf{1} + \mathrm{R}\,c})\,(\mathrm{inr}\,r)\,q \simeq_A p\,r$.
(c) follows by (b) and $\mathrm{split}\,(\mathsf{while}\,u\,a\,p) = \mathrm{inl}\,\langle c, \lambda r.q(r) * \lambda x.\mathsf{while}\,u\,x\,p\rangle$.

□

5 Conclusion

We have identified a need for a general and workable way of representing and reasoning about interactive programs in dependent type theory. We introduced in dependent type theory the notion of an I/O-tree, parameterised over a world, making essential use of type dependency. We gave it in two forms. The first breaks normalisation, but is conceptually simpler and suitable if one is tolerant of a programming language with 'bottom', or divergent programs. The second preserves normalisation. We called programs of this kind "normalising I/O-programs". We introduced an equality relation identifying behaviourally indistinguishable programs and showed that the monad laws hold, modulo this equality. (For the normalising version these are Lemma 4.2 (a), 4.5 (a) and 4.4 (c)). We introduced while-loops in both versions and repeat-loops and redirect in the first version (and leave it as an interesting exercise to extend the last two constructions to the normalising version). In the non-normalising version the characteristic equations for while and repeat are fulfilled by definition, whereas in the normalising version we have shown them for while (Lemma 4.2 (c) and 4.5 (c)). We have characterised do as well in the latter version (Lemma 4.5 (b)).

In a future paper we will show how to move from one universe to another in the normalising version and explore what happens if C is a setoid with a specific equivalence relation. In addition we will introduce state-dependent I/O-programs, in which the set of commands available depends on the current state of knowledge about the world.

Appendix: Notations

In the paper we do not distinguish between Σ and Π-type on the logical framework level and as set-constructions. The empty set is denoted by $\mathbf{0}$, the set containing one element by $\mathbf{1}$ (with element $\bullet$). The set of natural numbers is denoted by $\mathbb{N}$. The injections for the disjoint union $A + B$ of sets A and B are written $\mathrm{inl} : A \to (A + B)$, $\mathrm{inr} : B \to (A + B)$. The elements of $\Sigma x : A.B$ are denoted by $\langle a, b\rangle$. The dependent function type (sometimes written as $\Pi x : A.B$) is denoted by $(x : A) \to B$, with abbreviations like $(x : A, y : B) \to C$ for $(x : A) \to (y : B) \to C$, $(x : A, B) \to C$ for $(x : A, y : B) \to C$ with y new, and $(x, y : A) \to B$ for $(x : A, y : A) \to B$. We use juxtaposition $(f\,a)$ for application, having a higher precedence than all other operators do, so that for example $f\,a = g\,b$ means $(f\,a) = (g\,b)$. The scope of variable-binding operators $\lambda x.$, $\forall x.$, $\exists x.$, $\Sigma x.$ is maximal (so $\lambda x.f\,a =_A b$ stands for $\lambda x.((f\,a) =_A b)$). Some functions

are represented as infix operators, writing some of the first few arguments as indices. (For instance we write $p *_{A,B} q$ for $(* \, A \, B \, p \, q)$.) Arguments that are written as indices are often omitted. We will omit the type in equality judgements, writing $r = s$ instead of $r = s : A$. An equation sign $=$ without indices denotes definitional equality, whereas we write $r =_A s$ (never omitting the A) for equality types (which are actually sets). The intensional equality has introduction rule $\mathrm{ref} : (A : \mathrm{Set}, a : A) \to a =_A a$ expressing reflexivity, and elimination rule $\mathrm{J} : (A : \mathrm{Set}, B : A \to \mathrm{Set}, a, a' : A, p : (a =_A a'), B\, a) \to B\, a'$, which corresponds to the second equality axiom: from $a =_A a'$ and $B\, a$ we can conclude $B\, a'$. The equality rule is $\mathrm{J}_A \, B \, a \, a \, \mathrm{ref}^A_a \, b = b$. Note that with extensional equality J could be defined trivially as $\lambda A, B, a, a', p, b.\, b$.

References

1. L. Augustsson. Cayenne — a language with dependent types. In *Proc. of the International Conference on Functional Programming (ICFP'98)*. ACM Press, September 1998.
2. L. Hallnäs. An intensional characterization of the largest bisimulation. *Theoretical Computer Science*, 53:335–343, 1987.
3. P. Hancock and A. Setzer. The IO monad in dependent type theory. DTP'99, `http://www.md.chalmers.se/Cs/Research/Semantics/APPSEM/dtp99/proceedings.html`, 1999.
4. J. Hughes. The design of a pretty-printing library. In J. Jeuring and E. Meijer, editors, *Advanced Functional Programming*, volume 925 of *LNCS*, pages 53–93. Springer, 1995.
5. I. Lindström. A construction of non-well-founded sets within Martin-Löf's type theory. *Journal of Symbolic Logic*, 54(1):57–64, 1989.
6. P. Martin-Löf. Constructive mathematics and computer programming. In J. L. Cohen, J. Łoś, H. Pfeiffer, and K.-D. Podewski, editors, *Proceedings 6th Intl. Congress on Logic, Methodology and Philosophy of Science, Hannover, FRG, 22–29 Aug 1979*, pages 153–175. North Holland, Amsterdam, 1982.
7. P. Martin-Löf. *Intuitionistic Type Theory*, volume 1 of *Studies in Proof Theory: Lecture Notes*. Bibliopolis, Napoli, 1984.
8. E. Moggi. Notions of computation and monads. *Information and Computation*, 93(1):55–92, 1991.
9. B. Nordström, K. Petersson, and J. M. Smith. *Programming in Martin-Löf's Type Theory: An Introduction*. Clarendon Press, Oxford, 1990.
10. S. L. Peyton Jones and P. Wadler. Imperative functional programming. In *20'th ACM Symposium on Principles of Programming Languages*, Charlotte, North Carolina, January 1993.
11. A. Setzer. Well-ordering proofs for Martin-Löf type theory. *Annals of Pure and Applied Logic*, 92:113 – 159, 1998.
12. P. Wadler. The essence of functional programming. In *19'th Symposium on Principles of Programming Languages, Albuquerque*, volume 19. ACM Press, January 1992.
13. P. Wadler. Monads for functional programming. In J. Jeuring and E. Meijer, editors, *Advanced Functional Programming*, volume 925 of *LNCS*. Springer Verlag, 1995.

Modal Satisfiability Is in Deterministic Linear Space

Edith Hemaspaandra*

Department of Computer Science,
Rochester Institute of Technology, Rochester, NY 14623, USA.
eh@cs.rit.edu

Abstract. In recent years, there has been a lot of interest in analyzing the space requirements for modal logics. In this paper, we prove that modal satisfiability is in deterministic linear space. This improves the best previously-known $O(n \log n)$ bound and it is the first linear space result in this area.

1 Introduction

In 1977, Ladner [5] showed that the modal satisfiability problems for K, T, and S4 are PSPACE-complete. His decision procedures for K, T, and S4 use deterministic space $O(n^2)$, $O(n^3)$, and $O(n^4)$ respectively. Since the goal of his paper was to prove PSPACE-completeness, it is not surprising that these upper bounds are not optimal.

Hudelmaier [6] proved that K and T satisfiability can be decided in deterministic space $O(n \log n)$ and that S4 satisfiability can be decided in deterministic space $O(n^2 \log n)$. A deterministic $O(n^2 \log n)$ space upper bound for K4 was proven by Vigano [12]. The $O(n^2 \log n)$ bounds for K4 and S4 satisfiability were recently improved to $O(n \log n)$ deterministic space upper bounds by Nguyen [7]. See also Basin et al. [1,2] for uniform methods to obtain space upper bounds for non-classical logics. The first question that this paper addresses is whether these ubiquitous $O(n \log n)$ bounds are optimal.

It is interesting to note that all these papers use proof-theoretic methods rather than semantic methods. This is unusual, since semantic methods are much more common in proving complexity results for logics. The second question that this paper addresses is whether semantic methods are unsuitable for proving precise space bounds.

In this paper, we provide a negative answer to both these questions. We will show that modal satisfiability (i.e., K satisfiability) is in deterministic linear space, using purely semantic arguments.

The paper is organized as follows. In the next section, we give some basic background and terminology about modal logic and about space complexity. In Section 3, we give a quadratic space algorithm for modal satisfiability. This algorithm will be the starting point for the non-deterministic linear space algorithm

* Supported in part by grant NSF-INT-9815095/DAAD-315-PPP-gü-ab.

P. Clote and H. Schwichtenberg (Eds.): CSL 2000, LNCS 1862, pp. 332–342, 2000.

from Section 4, which in Section 5 will be converted into a deterministic linear space algorithm.

2 Preliminaries

2.1 Modal Logic

We will briefly review syntax, Kripke semantics, and some basic terminology for modal logic.

Syntax

The set of (modal) formulas is inductively defined as follows.

- p is a formula for every propositional variable p,
- if ϕ and ψ are formulas, then so are $\phi \wedge \psi$ and $\neg\phi$, and
- if ϕ is a formula, then $\Box\phi$ is a formula.

The *modal depth* of a formula ϕ (denoted by $md(\phi)$) is the depth of nesting of the modal operator $\Box$. Formally,

- $md(p) = 0$ for every propositional variable p
- $md(\neg\phi) = md(\phi)$
- $md(\phi \wedge \psi) = \max(md(\phi), md(\psi))$
- $md(\Box\phi) = md(\phi) + 1$

Semantics

A (Kripke) *model* is of the form $M = <W, R, \pi>$ such that W is a non-empty set of possible worlds, R is a binary relation on W called the accessibility relation, and π is a valuation, i.e., a function from the set of propositional variables to $Pow(W)$. $\pi(p)$ is the set of worlds in which p is true. For a formula ϕ, we will write $M, w \models \phi$ for ϕ is *true/satisfied at* w *in* M. The truth relation $\models$ is inductively defined on the structure of ϕ in the following way.

- $M, w \models p$ iff $w \in \pi(p)$ for p a propositional variable.
- $M, w \models \neg\phi$ iff not $M, w \models \phi$.
- $M, w \models \phi \wedge \psi$ iff $M, w \models \phi$ and $M, w \models \psi$.
- $M, w \models \Box\phi$ iff $\forall w' \in W[wRw' \Rightarrow M, w' \models \phi]$.

A modal formula ϕ is satisfiable (K satisfiable) if and only if there exists a model $M = <W, R, \pi>$ and a world $w \in W$ such that $M, w \models \phi$. It is easy to see that a formula ϕ is satisfiable if and only if it is satisfiable in the root of a tree.

Since we want to restrict the amount of space needed, we will, during the construction of the model, only look at formulas that are relevant in each constructed world. Of course, the only formulas that are relevant form a subset of the set of subformulas of ϕ, usually denoted by $Cl(\phi)$. But we need to be more precise than that. In order to do so, we introduce the following definition.

Definition 1. *Define* $Cl(\phi, d)$ *where* ϕ *is modal formula and depth* $d \geq 0$ *as follows:*

1. $\phi \in Cl(\phi, 0)$
2. *If* $\neg\psi \in Cl(\phi, d)$*, then* $\psi \in Cl(\phi, d)$
3. *If* $\psi \wedge \xi \in Cl(\phi, d)$*, then* $\psi \in Cl(\phi, d)$ *and* $\xi \in Cl(\phi, d)$
4. *If* $\Box\psi \in Cl(\phi, d)$*, then* $\psi \in Cl(\phi, d+1)$

As mentioned before, ϕ is satisfiable if and only if ϕ is satisfiable in the root of a tree. If a model M is a tree, and the root of the tree satisfies ϕ, then for every world w at depth d, the only formulas that are relevant are those in $Cl(\phi, d)$.

2.2 Space Complexity

In this subsection, we review some well-known relationships between time and space classes.

1. P $\subseteq$ NP $\subseteq$ PSPACE = NPSPACE $\subseteq$ EXPTIME.
   The equality PSPACE = NPSPACE follows from Savitch's theorem [9]. It is known that P is a strict subset of EXPTIME, but it is not known which of the inclusions are strict.
2. For linear time/space, the inclusions are as follows:
   DTIME $(O(n)) \subsetneq$ NTIME $(O(n)) \subseteq$ DSPACE$(O(n)) \subseteq$ NSPACE$(O(n)) \subseteq$ DTIME$(2^{O(n)})$.
   DTIME $(O(n)) \subsetneq$ NTIME $(O(n))$ is due to Paul, Pippinger, Szemerédy, and Trotter [8]. The strictness of the other inclusions is unknown. In addition, Savitch's theorem gives NSPACE $(O(n)) \subseteq$ DSPACE $(O(n^2))$.
3. Stearns, Hartmanis, and Lewis [10] proved the following hierarchy theorem for deterministic space: If $S_2(n)$ is a space-constructible function, $S_1(n) \leq S_2(n)$ for all n, and $\inf_{n\to\infty} \frac{S_1(n)}{S_2(n)} = 0$, then DSPACE$(S_1(n)) \subsetneq$ DSPACE$(S_2(n))$ [10].

PSPACE features prominently in the complexity of modal satisfiability problems. For example, the satisfiability problems for K, T, K4, and S4 are all PSPACE-complete [5] and so are their multi-modal analogues and multi-modal S5 [4].

3 Satisfiability in Quadratic Space

In this section, we will give a quadratic space algorithm for modal satisfiability. This algorithm forms the basis for the linear space algorithm that will be given in the next section. In addition, it introduces some notation, tools, and observations that will be built upon in the next sections.

It is easy to show with induction on depth d, that for all $\Gamma \subseteq Cl(\phi, d)$, Γ is a maximal satisfiable subset of $Cl(\phi, d)$ if and only if $MaxSat(\Gamma, d)$ is true, where $MaxSat(\Gamma, d)$ is recursively defined as follows. $MaxSat(\Gamma, d)$ is true if and only if

1. ($\psi \notin \Gamma$ iff $\neg\psi \in \Gamma$) for all $\neg\psi$ in $Cl(\phi, d)$,
2. ($\psi \wedge \xi \in \Gamma$ iff $\psi \in \Gamma$ and $\xi \in \Gamma$) for all $\psi \wedge \xi \in Cl(\phi, d)$, and
3. for all $\Box\psi$ in $Cl(\phi, d) \setminus \Gamma$, there exists a set $\Gamma_{\neg\psi} \subseteq Cl(\phi, d+1)$ such that
   (a) $\psi \notin \Gamma_{\neg\psi}$,
   (b) for all $\Box\xi \in \Gamma$, $\xi \in \Gamma_{\neg\psi}$,
   (c) $MaxSat(\Gamma_{\neg\psi}, d+1)$.

This definition is the starting point for our space efficient algorithms for modal satisfiability, since ϕ is satisfiable if and only if there exists a set of formulas $\Gamma \subseteq Cl(\phi, 0)$ such that $\phi \in \Gamma$ and $MaxSat(\Gamma, 0)$.

This definition is close to Ladner's tableau construction [5]. The main differences are that we have replaced the tableau rules for the propositional part by one nondeterministic step, and that we are explicit about the set of formulas relevant at each recursive depth. The definition is even closer to Vardi's construction [11]. The main difference is that we are explicit about the set of formulas relevant at each recursive depth.

Of course, the recursive definition of *MaxSat* given above is not quite an algorithm, but that is easy to fix. Here is a nondeterministic algorithmic version of *MaxSat*, closely related to Vardi's alternating polynomial time algorithm [11]. (Alternating polynomial time = PSPACE [3].) The conversion from definition to algorithm is straightforward, and it will make the transition to the linear space algorithm easier.

For all $\Gamma \subseteq Cl(\phi, d)$, the algorithm will accept $MaxSat(\Gamma, d)$ if and only if Γ is a maximal satisfiable subset of $Cl(\phi, d)$.

$MaxSat(\Gamma, d)$:
For every $\phi' \in Cl(\phi, d)$,

1. if $\phi' = \neg\psi$ and *not* [$\psi \notin \Gamma$ iff $\neg\psi \in \Gamma$], then reject
2. if $\phi' = \psi \wedge \xi$ and *not* [$\psi \wedge \xi \in \Gamma$ iff $\psi \in \Gamma$ and $\xi \in \Gamma$], then reject
3. if $\phi' = \Box\psi$ and $\phi' \notin \Gamma$, then guess a set $\Gamma_{\neg\psi} \subseteq Cl(\phi, d+1)$ such that
   (a) $\psi \notin \Gamma_{\neg\psi}$,
   (b) for all $\Box\xi \in \Gamma$, $\xi \in \Gamma_{\neg\psi}$, and
   (c) $MaxSat(\Gamma_{\neg\psi}, d+1)$ accepts.

Accept (that is, if the algorithm got though the loop without rejecting, then accept).

We will now briefly and informally analyze the space required for this algorithm. A more formal analysis of the linear space algorithms will be given in the next sections.

To analyze the space required by the algorithm, first note that the maximum number of nested recursive calls is $md(\phi) + 1$, since $Cl(\phi, md(\phi) + 1) = \emptyset$. The amount of space required for each call to *MaxSat* without the recursive calls is dominated by the space required to store $\Gamma_{\neg\psi}$. It is well-known that every subset of subformulas of ϕ can be represented as a bitstring of length $|\phi|$, since every position in ϕ corresponds to at most one subformula of ϕ, and every subformula

of ϕ corresponds to at least one position in ϕ. (We leave the exact details of the representation and implementation for the next section.)

These observations lead to a quadratic nondeterministic space upper bound. In general, this would give a quartic deterministic space upper bound via Savitch's theorem [9]. However, in this algorithm, the nondeterminism is used in a rather restricted way. The only nondeterminism in the algorithm is in step 3: "guess a set $\Gamma_{\neg\psi} \subseteq Cl(\phi, d+1)$ such that ..."

We can remove this nondeterminism without increase in space in the following way. Replace step 3 by

3. if $\phi' = \Box\psi$ and $\phi' \notin \Gamma$, then cycle through all sets $\Gamma_{\neg\psi} \subseteq Cl(\phi, d+1)$. For each of these subsets, check if
   (a) $\psi \notin \Gamma_{\neg\psi}$,
   (b) for all $\Box\xi \in \Gamma$, $\xi \in \Gamma_{\neg\psi}$, and
   (c) $MaxSat(\Gamma_{\neg\psi}, d+1)$ accepts.
   If we find that one of the sets $\Gamma_{\neg\psi} \subseteq Cl(\phi, d+1)$ satisfies these three requirements, then proceed with the algorithm. Otherwise, reject.

We will leave the details about how to deterministically cycle though all subsets of $Cl(\phi, d+1)$ to Section 5.

4 Satisfiability in Nondeterministic Linear Space

The quadratic deterministic space bound from the previous section ties Ladner's bound, which is not surprising, since our algorithm is close to Ladner's. How can we do better? In the analysis, we looked at the space used for each nested recursive call separately. In order to improve the space efficiency, we will have to combine the space used at the different recursion depths. Since recursion in combination with global variables is hard to follow, we will give an iterative version for satisfiability that is based on *MaxSat* and work from there.

One problem is to simulate the multiple recursive calls $MaxSat(\Gamma_{\neg\psi}, d+1)$ without using too much space. For example, if we keep track of the number of recursive calls made at each depth, we need space $n \log n$, and this is too much. Our solution is the following. For each depth d, assume an ordering on the formulas in $Cl(\phi, d)$. It then suffices to keep track of the formula currently being processed. We will call this formula $curForm(d)$. One might think that we will then need $\log n$ bits at each depth d to keep track of $curForm(d)$, in which case we would have gained nothing. But we will show in the sequel that all these formulas together can be combined into one length n bitstring.

The remainder of the simulation is fairly straightforward. Depth d corresponds to d in *MaxSat* and $\Gamma(d)$ corresponds to Γ at depth d in *MaxSat*. We will use *newWorld* to denote that a new world is being built, i.e., all formulas in $Cl(\phi, d)$ still have to be processed.

Here is the non-recursive and nondeterministic algorithm to determine if ϕ is satisfiable.

$d := 0$; $\quad newWorld := true$
guess $\Gamma(0) \subseteq Cl(\phi, 0)$ such that $\phi \in \Gamma(0)$
`while` $d \geq 0$ `do`
  `if` not $newWorld$ and $curForm(d)$ is the last formula in $Cl(\phi, d)$ `then`
    $d := d - 1$
  `else`
    `if` $newWorld$ `then`
      $curForm(d) :=$ the first formula in $Cl(\phi, d)$
      $newWorld := false$
    `else`
      $curForm(d) :=$ the next formula in $Cl(\phi, d)$
    `if` $curForm(d) = \neg\psi$ and $not\ [\psi \notin \Gamma(d)$ iff $\neg\psi \in \Gamma(d)]$ `then reject`
    `if` $curForm(d) = \psi \wedge \xi$ and
      $not\ [\psi \wedge \xi \in \Gamma(d)$ iff $\psi \in \Gamma(d)$ and $\xi \in \Gamma(d)]$ `then reject`
    `if` $curForm(d) = \Box\psi$ and $curForm(d) \notin \Gamma(d)$ `then`
      guess a set $\Gamma(d+1) \subseteq Cl(\phi, d+1)$ such that
        $\psi \notin \Gamma(d+1)$ and for all $\Box\xi \in \Gamma(d)$, $\xi \in \Gamma(d+1)$
      $d := d + 1$
      $newWorld := true$
`accept`

The space used by this algorithm depends on the implementation of Γ and *curForm*.

In the previous section, we mentioned that every subset of subformulas of ϕ can be represented as a bitstring of length $|\phi|$, since every position in ϕ corresponds to at most one subformula of ϕ, and every subformula of ϕ corresponds to at least one position in ϕ. We can store even more information in such a bitstring, since every position in ϕ corresponds to at most one *occurrence* of a subformula of ϕ, and every *occurrence* of a subformula of ϕ corresponds to exactly one position in ϕ. Since the sets of occurrences of formulas in $Cl(\phi, d)$ and $Cl(\phi, d')$ in ϕ are disjoint for all $d \neq d'$, this implies that we can represent the sequence of sets $\Gamma(0), \Gamma(1), \Gamma(2), \ldots$ by a length $|\phi|$ bitstring and also that we can represent $curForm(0), curForm(1), curForm(2), \ldots$ by a length $|\phi|$ bitstring.

This is looking good, since we are now using linear space to represent all relevant information of the algorithm. It remains to give the exact details of the representation and to show that we can encode and decode the relevant information into and from our representation without using more than linear space.

We will start with the definition of the representation. For ϕ a formula and $1 \leq i \leq |\phi|$ such that the ith symbol in ϕ (denoted by $\phi[i]$) is not a parenthesis, let ϕ_i be the ϕ subformula with $\phi[i]$ as main connective. The depth of i in ϕ, written as $depth_\phi(i)$ (or simply as $depth(i)$ if ϕ is clear from context) is defined as the modal nesting depth of the occurrence of the ϕ subformula with as main connective the ith symbol in ϕ.

The sequence of sets $\Gamma(0)$, $\Gamma(1)$, $\Gamma(2)$, ... will be encoded by bitstring Γ of length $|\phi|$ as follows.

- For all i, if $\phi[i]$ is a parenthesis, then $\Gamma[i] = 0$.
- If $\phi_i = \phi_j$ and $depth(i) = depth(j)$, then $\Gamma[i] = \Gamma[j]$.
- $\Gamma(d) = \{\psi_i \mid \Gamma[i] = 1 \text{ and } depth(i) = d\}$.

The sequence of currently active formulas $curForm(0)$, $curForm(1)$, $curForm(2)$, ... will be encoded by bitstring $curForm$ of length $|\phi|$ as follows.

- For all i, if $\phi[i]$ is a parenthesis, then $curForm[i] = 0$.
- If $curForm(d)$ is defined, then $curForm(d) = \phi_i$, where i is the unique i such that $curForm[i] = 1$ and $depth(i) = d$.
- If $curForm(d)$ is undefined, then there is no i such that $curForm[i] = 1$ and $depth(i) = d$.

It is easy to see that given an index i, we can compute ϕ_i and $depth(i)$ in linear space, for example by a simple modification of the standard infix-to-postfix conversion algorithm.

Now look carefully at the satisfiability algorithm. It is not hard to see that the whole algorithm can be implemented in linear space. (See below for the implementation of the relevant parts of the algorithm.)

Implementation of the Relevant Parts of the Algorithm

Computing $curForm(d)$:
```
for i := 1 to |φ| do
    if curForm[i] = 1 and depth(i) = d then
        curForm(d) := φ_i
```

Setting $curForm(d)$ to the next formula in $Cl(\phi, d)$:
```
for i := 1 to |φ| do
    if curForm[i] = 1 and depth(i) = d then break
curForm[i] := 0
for j := i + 1 to |φ| do
    if depth(j) = d then curForm[j] := 1; break
```

Checking if $\psi \in \Gamma(d)$:
```
for i := 1 to |φ| do
    if depth(i) = d and φ_i = ψ then
        ψ ∈ Γ(d) if and only if Γ[i] = 1
```

Guessing a set $\Gamma(d) \subseteq Cl(\phi, d)$:
```
for i := 1 to |φ| do
    if depth(i) = d then
        new := true
        for j := 1 to i − 1 do
            if depth(j) = d and φ_i = φ_j then
                new := false;   Γ[i] := Γ[j];   break
        if new then nondeterministically set Γ[i] to 0 or 1
```

5 Satisfiability in Deterministic Linear Space

In this section, we will show how to remove the nondeterminism from the nondeterministic linear space algorithm from Section 4, without increasing the amount of space used. Just as in Section 3, note that the nondeterminism in the algorithm from Section 4 is used in a restricted way, namely in guessing a subset of $Cl(\phi, d)$ such that certain properties are satisfied. As mentioned in Section 3, we can remove this nondeterminism without increase in space by cycling through all possible subsets of $Cl(\phi, d)$.

The following deterministic version of the algorithm from Section 4 algorithm makes this more precise. Implementation details of the new parts will follow.

$d := 0$; $reject := false$; $newWorld := true$
$\Gamma(0) :=$ the first subset of $Cl(\phi, 0)$
while $d \geq 0$ **do**
  if *reject* **then**
    if $\Gamma(d)$ is the last subset of $Cl(\phi, d)$ **then**
      if $d = 0$ **then reject else** $d := d - 1$
    else
      $\Gamma(d) :=$ the next subset of $Cl(\phi, d)$
      $reject := false$
      $newWorld := true$
  elseif not *newWorld* and $curForm(d)$ is the last formula in $Cl(\phi, d)$ **then**
    $d := d - 1$
  else
    if *newWorld* **then**
      if $d = 0$ and $\phi \notin \Gamma(0)$ **then** $reject := true$
      // if $curForm(d-1) = \Box\psi$, then we need a witness for $\Box\psi \notin \Gamma(d-1)$
      if $d > 0$ and $curForm(d-1) = \Box\psi$ and $\psi \in \Gamma(d)$ **then** $reject := true$
      if $d > 0$ and for some ξ, $\Box\xi \in \Gamma(d-1)$ and $\xi \notin \Gamma(d)$ **then**
        $reject := true$
    if not *reject* **then**
      if *newWorld* **then**
        $curForm(d) :=$ the first formula in $Cl(\phi, d)$
        $newWorld := false$
      else
        $curForm(d) :=$ the next formula in $Cl(\phi, d)$
      if $curForm(d) = \neg\psi$ and *not* $[\psi \notin \Gamma(d)$ iff $\neg\psi \in \Gamma(d)]$ **then**
        $reject :=$ *true*.
      if $curForm(d) = \psi \wedge \xi$ and *not* $[\psi \wedge \xi \in \Gamma(d)$ iff
        $\psi \in \Gamma(d)$ and $\xi \in \Gamma(d)]$ **then** $reject :=$ *true*.
      if $curForm(d) = \Box\psi$ and $curForm(d) \notin \Gamma(d)$ **then**
        $\Gamma(d+1) :=$ the first subset of $Cl(\phi, d+1)$
        $d := d + 1$
        $newWorld := true$
accept

We will use *newWorld* to denote that $\Gamma(d)$ is a new subset of $Cl(\phi, d)$. If *newWorld* is true, then we need to verify the desired properties of $\Gamma(d)$. Since we are simulating all possible nondeterministic choices of sets $\Gamma(d)$, we will use a variable *reject* that will be true if the current choice of Γ's rejects. If *reject* is true, we need to proceed to the next possible choice of Γ's.

We will use the same encoding as in the previous section. It should be clear that we are using deterministic linear space to keep track of all the relevant information in the algorithm. It remains to show that we can implement the new steps in the algorithm in linear space. This proves the main result of this paper.

Theorem 1. *Modal satisfiability is in deterministic linear space.*

Implementation of the New Parts of the Algorithm

Setting $\Gamma(d)$ to the first subset of $Cl(\phi, d)$, that is, setting $\Gamma(d)$ to $\emptyset$:

```
for i := 1 to |φ| do
    if depth(i) = d then
        Γ[i] := 0
```

Setting *reject* to *true* if for some ξ, $\Box\xi \in \Gamma(d-1)$ and $\xi \notin \Gamma(d)$:

```
for i := 1 to |φ| do
    if depth(i) = d − 1 and φ_i = □ξ and Γ[i] = 1 then
        for j := 1 to |φ| do
            if depth(j) = d and φ_j = ξ and Γ[j] = 0 then
                reject := true
```

It remains to show how to set $\Gamma(d)$ to the next subset of $Cl(\phi, d)$ and how to detect if $\Gamma(d)$ is the last subset of $Cl(\phi, d)$. Basically, we will view the bits $\Gamma[i]$ such that $depth(i) = d$ as a binary number, and set these bits to the next binary number, that is, we traverse the bits from right to left, changing every 1 to a until we see the first 0. Then change that 0 to a 1. However, we have to ensure that bitstring Γ properly encodes $\Gamma(d)$, that is, if $\phi_i = \phi_j$ and $depth(i) = depth(j) = d$, then $\Gamma[i] = \Gamma[j]$. We will repeat computing the next binary number until this is the case.

```
repeat
    found := false
    for i := |φ| downto 1 do
        if depth(i) = d then
            if Γ[i] = 1 then
                Γ[i] := 0
            else
                Γ[i] := 1
                found := true
                break
```

```
    if found then
        // check if Γ properly encodes Γ(d)
        correct := true
        for i := 1 to |φ| do
            for j := 1 to |φ| do
                if depth(i) = depth(j) = d and φ_i = φ_j and Γ[i] ≠ Γ[j] then
                    correct := false; break
until correct or not found
if not found then
    Γ(d) was the last subset of Cl(φ, d)
else
    Γ(d) was set to the next subset of Cl(φ, d).
```

6 What about Other Modal Logics?

In the linear space encoding of the algorithms for modal satisfiability, we crucially used the fact that every satisfiable formula is satisfiable in the root of a tree and that every world in the satisfying model is at a unique distance from the root. It is easy to see that, using the methods from this paper, one can obtain deterministic linear space upper bounds for, for example, satisfiability with respect to those models where every world has at most k successors, and for the disjoint union of any number of K logics. (This satisfiability problem was shown to be PSPACE-complete in [4].)

On the other hand, the constructions from this paper don't directly apply to satisfiability with respect to all reflexive models (T satisfiability), or transitive models (K4 satisfiability), since in these cases worlds will not be at a unique distance from the root. Without this property, we are not able to encode all relevant formulas on a branch of the model in one linear length bitstring.

However, we can construct satisfying K4 (transitive) and S4 (reflexive and transitive) models in quadratic space. In addition, we can keep track of all currently active $\Box$ formulas as one length n bitstring, as in the construction for *curForm* in this paper. This will lead to an $O(n \log n)$ space upper bound for these two logics. The details of these constructions are tedious, and these $O(n \log n)$ bounds only tie the best-known bounds [6,7], so we will not go into details. However, it does show that our methods are more widely applicable.

Acknowledgment

I would like to thank anonymous referees for useful comments and suggestions.

References

1. D. Basin, S. Matthews, and L. Viganò. A new method for bounding the complexity of modal logics. In *Proceedings of KGC'97*, pp. 89–102. Springer, LNCS 1289, 1997.

2. D. Basin and L. Viganò. A recipe for the complexity analysis of non-classical logics. In *Proceedings of FroCoS'98*, pp. 57–75. Wiley, Studies in Logic and Computation 7, 2000.
3. A.K. Chandra, D. Kozen, and L.J. Stockmeyer. Alternation. *Journal of the ACM*, 28, pp. 114–133, 1981.
4. J.Y. Halpern and Y. Moses. A guide to completeness and complexity for modal logics of knowledge and belief. *Artificial Intelligence*, 54, pp. 319–379, 1992.
5. R. Ladner. The computational complexity of provability in systems of modal propositional logic. *SIAM Journal on Computing*, 6(3), pp. 467–480, 1977.
6. J. Hudelmaier. Improved decision procedures for the modal logics K, T and S4. In *Proceedings of CSL'95*, pp. 320–334. Springer, LNCS 1092, 1996.
7. L.A. Nguyen. A new space bound for the modal logics K4, KD4, and S4. In *Proceedings of MFCS'99*, pp. 321–331. Springer, LNCS 1672, 1999.
8. W.J. Paul, N. Pippinger, E. Szemerédy, and W.T. Trotter. On determinism versus non-determinism and related problems. In *Proceedings of FOCS'83*, pp. 429–438, 1983.
9. W. J. Savitch. Relationships between nondeterministic and deterministic tape complexities, *JCSS*, 4, pp. 177-192, 1970.
10. R.E. Stearns, J. Hartmanis, and P.M. Lewis II. Hierarchies of memory limited computations. In *Proceedings of FOCS'65*, pp. 179–190, 1965.
11. M. Y. Vardi. On the complexity of epistemic reasoning. In *Proceedings of LICS'89*, pp. 243–252, 1989.
12. L. Viganò. *A framework for non-classical logics*. PhD thesis, Universität des Saarlandes, Saarbrücken, Germany, 1997.

Logic Programming and Co-inductive Definitions

Mathieu Jaume

Cermics –École Nationale des Ponts et Chaussées
Champs sur Marne 77455 Marne-La-Vallee Cedex 2, France
jaume@cermics.enpc.fr

Abstract. This paper aims to define a complete semantics for a class of non-terminating logic programs. Standard approaches to deal with this problem consist in concentrating on programs where infinite derivations can be seen as computing, in the limit, some "infinite object". This is usually done by extending the domain of computation with infinite elements and then defining the meaning of programs in terms of greatest fixpoints. The main drawback of these approaches is that the semantics defined is not complete. The approach considered here is exactly the opposite. We concentrate on the infinite derivations that do not compute an infinite term: this paper studies the operational counterpart of the greatest fixpoint of the one-step-inference operator for the $\mathcal{C}$-semantics. The main result is that such fixpoint corresponds to the set of atoms that have a non-failing fair derivation with the additional property that complete information over a variable is obtained after finitely many steps.

1 Introduction – Motivations

In computer science, termination of programs is a traditional requirement. Logic programming does not escape from this influence and there exist many works about termination of logic programs. However, infinite behaviour of programs can be useful to model some situations and the study of nonterminating "computations" has received an increasing interest in the context of many programming paradigms: λ-calculus, rewrite systems, logic programming, concurrent constraint programming [4] ... In this paper, we focus on *reactive logic programs* (i.e., definite logic programs for which the behaviours of interest are both termninating and non-termninating ones). In the field of logic programming, infinite SLD-derivations can be useful to model the infinite computation of an infinite object. As a typical example, with the program $P = \{\mathsf{LN}(x, [x|l]) \leftarrow \mathsf{LN}(S(x), l)\}$ we can obtain, from the query $\mathsf{LN}(k, l_0)$, an infinite derivation computing at every step a better approximation of the second argument. The "final result" is the "limit" of the sequence of approximations and corresponds to the infinite sequence of integers starting from k. However, there exist infinite derivations which do not compute an infinite object. Such derivations can be useful to model a certain class of infinite processes. For example, let us consider the famous (simple) dining philosophers problem, introduced by Dijkstra as a model for resource sharing. In this problem, 3 philosophers P_1, P_2 and P_3 are sitting around a table in the

P. Clote and H. Schwichtenberg (Eds.): CSL 2000, LNCS 1862, pp. 343–355, 2000.

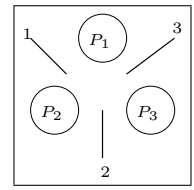


Fig. 1. The dining philosophers problem

center of which there is a plate of spaghetti. Between each philosopher and his (or her) neighbour there is exactly one fork. A philosopher requires two forks to be able to eat and since there are exactly as many forks as there are philosophers it is not possible for all philosophers to eat in the same time. We can describe this problem by the logic program containing the following clauses, expressing that a philosopher can take a fork, drop a fork, or eat:

$$\begin{array}{lll} p(0,x,y) \leftarrow p(1,x,y) & p(0,x,y) \leftarrow p(2,x,y) & p(x,0,y) \leftarrow p(x,3,y) \\ p(x,y,0) \leftarrow p(x,y,1) & p(x,0,y) \leftarrow p(x,2,y) & p(x,y,0) \leftarrow p(x,y,3) \\ p(1,x,y) \leftarrow p(0,x,y) & p(2,x,y) \leftarrow p(0,x,y) & p(x,3,y) \leftarrow p(x,y,0) \\ p(x,y,1) \leftarrow p(x,y,0) & p(x,2,y) \leftarrow p(x,0,y) & p(x,y,3) \leftarrow p(x,y,0) \\ \mathsf{eat}(1) \leftarrow p(1,x,1) & \mathsf{eat}(2) \leftarrow p(2,2,x) & \mathsf{eat}(3) \leftarrow p(x,3,3) \end{array} \quad (1)$$

where p is a 3-ary predicate whose three arguments describe respectively the state of the 3 forks (0 means the fork is free, $i \in \{1,2,3\}$ means the fork is taken by the philosopher P_i) and $\mathsf{eat}(i)$ means that the philosopher P_i can eat. With this program, every derivation from the query $\mathsf{eat}(i)$ is infinite and does not compute an infinite object. It just describes an infinite sequence of actions done by the philosophers[1].

The main approaches to assign some meaning to infinite derivations in "pure" logic programming occurring in the literature [1,7,8,15,16] concentrate on the aspects related to the semantics of infinite objects and to the models for logic programs which take them into account. In this setting, the relevant notion is the one of computation (in the limit) of an infinite object. The sense of a "useful" infinite derivation is given by the notion of *atom computed at infinity* (i.e., an infinite atom A such that there exists a finite atom from which there exists an infinite derivation which "computes at infinity" A). However, these approaches are not satisfactory since if we consider a greatest fixpoint semantics over the domain of infinite terms, then programs like program

$$P = \{p(x) \leftarrow p(x)\} \quad (2)$$

have a non-empty denotation even if no atom can be computed at infinity with P. In fact, $p(f^\omega)$ is in the greatest fixpoint of the classical one-step-inference

[1] This behaviour may correspond to a livelock situation, however, our interest in this problem is different.

operator associated to P even if $p(f^\omega)$ is not computable by an infinite derivation. The construction of the greatest fixpoint does not reflect how the infinite terms are constructed during a derivation. Hence, either such semantics [7,8,16] are not complete, since there exist infinite atoms in the denotation of a program which are not computable by an infinite derivation, or the completeness is expressed as follows [1,10,11]:

$$A \in \mathsf{gfp}(T_P) \Leftrightarrow A \text{ is the root of a fair derivation}$$

where A is a possibly infinite atom. Nevertheless, in this case, this "completeness" result is obtained by allowing infinite terms in queries[2] which start SLD-derivations. For example, with program (2), we can consider the derivation:

$$p(f^\omega) \rightarrow_P \cdots \rightarrow_P p(f^\omega) \rightarrow_P \cdots \tag{3}$$

This requirement is clearly stated in [18]. Nevertheless, this does not correspond to the standard operational semantics of logic programs as defined in [16] and one may wonder how queries containing infinite terms can be given.

It is now well-known that standard semantics of logic programs can be expressed by purely proof-theoretic methods [9]. The most immediate way to give such a semantics is to consider clauses as inference rules, rather than logic formulas, and then a logic program as a formal system. From this point of view, the denotation of a program is the set of theorems which can be derived in this system. Within this framework, (co-)inductive definitions are a natural way to define the denotation of logic programs. Since, proof-theoretically, we can look at a clause $A \leftarrow B_1, \cdots, B_n$ as an introduction rule for A (or similarly as a constructor in an (co-)inductive definition), by following the Curry-Howard isomorphism, it is possible to represent clauses by constructors of a functional language and each proof can be viewed as a functional expression. Hence, in this paper, we focus on the correspondence between co-inductive definitions and logic programs (i.e., between proofs as functional expressions and proofs as SLD-derivations). This will lead us to define a sound and complete semantics for the subclass of *infinite derivations over the domain of finite terms* (i.e., infinite derivations which do not compute any infinite term). One may question about the interest of such derivations which are often outcast as meaningless. However, as we said, such derivations are useful to model infinite processes which do not compute an infinite object. Furthermore, it seems that incompleteness of usual approaches comes from these derivations and this work complements the knowledge we have in this area. Due to space limitations, most of proofs are incomplete or omitted here. They can be found in a research report [12]. The rest of the paper is organized as follows: the end of this section introduces the basic definitions and notations. Section 2, which can be omitted during the first reading, contains a (brief) discussion on the correspondence between SLD-derivations and proof terms, and last, section 3 presents a sound and complete semantics for infinite derivations over the domain of finite terms.

[2] In an implicit way, this implies the use of an adequate unification algorithm and the modification of the notion of atom computed at infinity.

Background and notations We assume here familiarity with the standard notions of (co-)inductive definitions and logic programming (Herbrand semantics and $\mathcal{C}$-semantics) as introduced in [2,6,14,16]. (Co-)inductive sets can be defined by some rules for generating elements of the set and by adding that an object is to be in the set only if it has been generated by applying these rules. Given a rule set Φ (a rule is written $e \leftarrow E$, where E is the set of premises, and e is the conclusion) a set A is said to be Φ-closed (resp. Φ-dense) if each rule in Φ whose premises are in A also has its conclusion in A (resp. if for every $a \in A$ there is a set $E \subseteq A$ such that $(a \leftarrow E) \in \Phi$). The set inductively (resp. co-inductively) defined by Φ, written $\mathsf{Ind}(\Phi)$ (resp. $\mathsf{CoInd}(\Phi)$), is defined by $\mathsf{Ind}(\Phi) = \cap\{A \mid A \text{ is } \Phi\text{-closed}\}$ (resp. $\mathsf{CoInd}(\Phi) = \cup\{A \mid A \text{ is } \Phi\text{-dense}\}$). These sets can also be expressed by using monotone operators: if Φ is a rule set, we may define a monotone operator $T_\Phi : 2^{\mathcal{B}} \to 2^{\mathcal{B}}$, where $\mathcal{B} = \cup_{e \leftarrow E \in \Phi}\{\{e\} \cup E\}$, by:

$$T_\Phi(A) = \{e \in \mathcal{B} \mid \exists\ e \leftarrow E \in \Phi, \quad E \subseteq A\} \tag{4}$$

and then $\mathsf{Ind}(\Phi) = \cap_{T_\Phi(A) \subseteq A} A = \mathsf{lfp}(T_\Phi)$ and $\mathsf{CoInd}(\Phi) = \cup_{A \subseteq T_\Phi(A)} A = \mathsf{gfp}(T_\Phi)$. Σ, Π and X denote respectively a set of function symbols, a set of predicate symbols, and a set of variable symbols. Elements of $T_\Sigma[X]$ are (finite) terms over $X \cup \Sigma$. A substitution θ is a mapping from X to $T_\Sigma[X]$ such that $\{x \mid x \neq \theta x\} = dom(\theta)$ is finite. $range(\theta)$ denotes the set $\{var(\theta x) \mid x \in dom(\theta)\}$. Composition of substitutions induces a preorder on substitutions ($\theta_1 \leq \theta_2 \Leftrightarrow \exists \mu, \mu\theta_1 = \theta_2$) and on expressions ($E_1 \leq E_2 \Leftrightarrow \exists \mu, \mu E_1 = E_2$). A renaming substitution is a mapping $\sigma\colon X \to X$ such that $\forall x, y \in dom(\sigma)$, $x \neq y \Rightarrow \sigma(x) \neq \sigma(y)$. A mgu is a minimal idempotent unifier. The preorder $\leq$ induces an equivalence relation $\approx$ (called variance): $E_1 \approx E_2$ iff there exist two renaming substitutions θ_1 and θ_2 such that $\theta_1 E_1 = E_2$ and $\theta_2 E_2 = E_1$. $At_{\Sigma,\Pi}[X]$ denotes the set of (finite) atoms. Given a clause $C \in P$, we write C^+ for its head and C^- for its body. An SLD-derivation with a program P is a possibly infinite sequence of transitions:

$$\underbrace{A_1, \cdots, A_k, \cdots, A_n}_{R} \overset{C,\theta}{\rightarrow}_P \underbrace{\theta(A_1, \cdots, A_{k-1}, B_1, \cdots, B_q, A_{k+1}, \cdots, A_n)}_{\theta R[k \leftarrow C^-]}$$

where θ is a mgu of C^+ and A_k and where C is a variant of a clause in P, whose body is $B_1, \cdots, B_q$. In an SLD-derivation from a goal R_0, the sequence of clauses $C_1, C_2, \cdots$ is such that[3]:

$$\forall i \geq 1 \quad var(C_i) \cap (\cup_{j<i} var(C_j) \cup var(R_0)) = \emptyset$$

As defined in [16], an SLD-derivation is fair if it is either failed or, for every atom B in the derivation, (some further instantiated version of) B is selected within a finite number of steps. Given an atom A (resp. a program P), we write $\lceil A \rceil$ (resp. $\lceil P \rceil$) to denote the set of (not necessarily ground) finite instances of A (i.e., of clauses in P). Furthermore P also denotes all the variants of clauses in P.

[3] As illustrated in [13], this renamning process is crucial and has been explicitely considered in our proofs. All the derivations considered here satisfy this requirement.

2 Logic Programs as (Co-)inductive Definitions

The fixed point semantics has long been used as a technical device. However, it corresponds to the "logic programs as inductive (co-)definitions" paradigm and can be considered as the logic program's intrinsic declarative content. Indeed, many properties of logic programs are similar to these enjoyed by inductive definitions. Recall that, as proved in [6], a $\mathcal{C}$-interpretation I (i.e., an upward closed subset of $At_{\Sigma,\Pi}[X]$) is a $\mathcal{C}$-model of P iff $T_P^{\mathcal{C}}(I) \subseteq I$ and the model intersection property allows to consider the least $\mathcal{C}$-model of P as the intersection of all $\mathcal{C}$-models of P. Since $T_P^{\mathcal{C}}$ is exactly the operator $T_{\lceil P \rceil}$ obtained from the rule set $\lceil P \rceil$, as described by (4), each $\mathcal{C}$-model of P is a $T_{\lceil P \rceil}$-closed set and, since $\mathsf{Ind}(T_{\lceil P \rceil})$ is defined as the intersection of all $T_{\lceil P \rceil}$-closed sets, we have $\mathcal{M}_P^{\mathcal{C}} = \mathsf{Ind}(\lceil P \rceil)$. Now, since the body of each clause contains a finite number of atoms, $T_{\lceil P \rceil}$ is continuous and we have the well-known result $\mathcal{M}_P^{\mathcal{C}} = \mathsf{lfp}(T_{\lceil P \rceil}) = T_{\lceil P \rceil}^{\uparrow\omega}$ which only follows from properties of inductive definitions: the least $\mathcal{C}$-model can be directly expressed by an inductive definition. This proof-theoretic approach is now well-known [9,17,5] and has been used to extend logic programming languages in order to increase the power of "pure" declarative programming. Now let us consider the "logic programs as co-inductive definitions" paradigm, from which the usual greatest fixpoint semantics is defined. The greatest fixpoint of the operator, defined over the completed Herbrand base (i.e., containing infinite atoms), associated to a program P, corresponds to the co-inductive set $\mathsf{CoInd}(\llbracket P \rrbracket)$, where $\llbracket P \rrbracket$ denotes all the ground instances of clauses occurring in P over the completed Herbrand base. Hence, incompleteness follows from programs like program (2) since for this program, the clause $p(f^\omega) \leftarrow p(f^\omega)$ is in $\llbracket P \rrbracket$ and therefore $\{p(f^\omega)\}$ is $\llbracket P \rrbracket$-dense (i.e., $T_{\llbracket P \rrbracket}$-dense) and then we have $p(f^\omega) \in \mathsf{CoInd}(\llbracket P \rrbracket)$. The incompleteness comes from the fact that clauses of $\llbracket P \rrbracket$ are expressed over a language richer than the language of programs and queries: by allowing infinite elements in queries, such an approach becomes complete (since we can obtain the derivation (3)).

In the context of type theory, T. Coquand [3], note that infinite objects can be constructively understood, without the consideration of partial elements or greatest fixed-points, through the explicit consideration of proof objects. As said in section 1, by following the proofs-as-programs principle, we can look at a clause as a constructor of a functional language and then each proof can be represented as a functional expression. Like in a programming language, such expressions can be defined by recursion which corresponds to proofs where the result proved is used recursively. Of course, this cannot be considered to be a valid proof in general, and has to satisfy the guardedness property: "[3] *in order to establish that a proposition ϕ follows from other propositions $\phi_1, \cdots, \phi_q$, it is enough to build a proof term e for it, using not only natural deduction, case analysis, and already proven lemmas, but also using the proposition we want to prove recursively, provided such a recursive call is guarded by introduction rules.*". Hence, by considering clauses as introduction rules, and since a clause is applied at each resolution step of a derivation, it is possible to establish a

correspondence between guarded (proof) terms in a co-inductive set and SLD-derivations (more formally, a term is said to be guarded (by constructors) if its definition is such that all the recursive calls of the definition are done after having explicitly mentioned which is (at least) the first rule to start building the element and such that no other functions apart from constructors are applied to recursive calls). Let us introduce two examples. With the program $P = \{p(x) \leftarrow p(f(x))\}$, we can obtain the derivation:

$$p(z) \overset{C}{\rightarrow}_P \underbrace{\left[\begin{smallmatrix} x_1 \\ f(x_1) \end{smallmatrix}\right] p(x_1) \overset{C}{\rightarrow}_P \cdots \overset{C}{\rightarrow}_P \left[\begin{smallmatrix} x_i \\ f^i(x_i) \end{smallmatrix}\right] p(x_i) \overset{C}{\rightarrow}_P \cdots}_{\text{proof of } \forall x_1\ p(f(x_1))}$$

which can be viewed as a proof of $\forall x\ p(x)$, since the term:

$$\pi{:=}\lambda z.C\left(z, \pi\left(\left[\begin{smallmatrix} z \\ f(z) \end{smallmatrix}\right] p(z)\right)\right)$$

is guarded by the clause (i.e., the rule) C and then defines for any z a proof of $p(z)$ which belongs to $\mathsf{CoInd}(\lceil P \rceil)$. The correspondence is immediate: the application of the constructor C corresponds to the first transition of the derivation, while the recursive call corresponds to the next ones (i.e., the derivation starting from the query $p(f(x_1))$ – a proof of $\forall x\, p(f(x))$). Such derivations correspond to co-inductive proofs. However, this correspondence cannot be observed for infinite SLD-derivations which compute infinite terms. Consider for example the program $P = \{p(f(x)) \leftarrow p(x)\}$ from which we can obtain the following infinite derivation computing the infinite term f^ω:

$$p(z) \overset{\left[\begin{smallmatrix} z \\ f(x_1) \end{smallmatrix}\right]}{\longrightarrow_P} p(x_1) \overset{\left[\begin{smallmatrix} x_1 \\ f(x_2) \end{smallmatrix}\right]}{\longrightarrow_P} \cdots \overset{\left[\begin{smallmatrix} x_{i-1} \\ f(x_i) \end{smallmatrix}\right]}{\longrightarrow_P} p(x_i) \overset{\left[\begin{smallmatrix} x_i \\ f(x_{i+1}) \end{smallmatrix}\right]}{\longrightarrow_P} \cdots$$

Such a derivation is both a computation (of the infinite term f^ω) and a proof that this infinite term is such that $p(f^\omega)$. However, the proof term of $p(f^\omega)$ is defined by:

$$\pi{:=}\ \mathsf{eq\_ind}(f(f^\omega), p, C(f^\omega, \pi), f^\omega, \ell_\omega)$$

where C is the clause in P, where ℓ_ω is a proof of $f^\omega = f(f^\omega)$[4], and where $\mathsf{eq\_ind}$ corresponds to Leibniz'equality:

$$\mathsf{eq\_ind} : \forall x \in E,\ \forall P \text{ predicate on } E,\ P(x) \Rightarrow \forall y \in E\ (y = x) \Rightarrow P(y)$$

Clearly, this proof term does not correspond to the infinite derivation computing the term f^ω. However, proof terms over finite objects do not use $\mathsf{eq\_ind}$ and can be viewed as definitions of the sequences of clauses used in the corresponding derivations.

[4] Note that f^ω is defined by a guarded by constructors definition ($f^\omega{:=}f(f^\omega)$) – possibly infinite terms are co-inductively defined with function symbols as constructors.

3 Infinite SLD-Derivations over the Domain of Finite Terms

Since the presence of infinite elements in the Herbrand base leads to incompleteness of the approaches based on greatest fixpoints, we focus in the following on infinite derivations which do not compute infinite terms.

3.1 Proof Trees and Fair Derivations

First, we define SLD-proofs (for the operational semantics based on SLD-resolution) and proof trees (for the declarative semantics based on greatest fixpoint methods) as follows:

Definition 1. *An **SLD-proof** is either a refutation or a fair infinite derivation.*

Definition 2. *Let Φ be a rule set over $\mathcal{B}$. A **proof tree** of $x \in \mathcal{B}$ for Φ is a possibly infinite tree T such that x is the root of T, and for every node z occurring in T with $z_1, \cdots, z_n$ as children, there exists a rule $z \leftarrow z_1, \cdots, z_n \in \Phi$ (if z is a leaf, there exists a rule $z \leftarrow \in \Phi$).*

In the following, we say that T is a *partial proof tree* if T is a proof tree whose leaves do not necessarily correspond to a (unit) rule. We have the following well-known lemma.

Lemma 1. *$x \in \mathsf{CoInd}(\Phi)$ iff x is the root of a proof tree for Φ.*

Furthermore, the proof tree is finite iff $x \in \mathsf{Ind}(\Phi)$ (for finitary Φ).

In the next subsection, in order to prove the completeness result, we will need to be able to "translate" a proof tree into an SLD-derivation. The following lemma shows how this translation can be done.

Lemma 2. *Given a rule set Φ and an atom $A \in \mathsf{CoInd}(\Phi)$, there exists an SLD-proof from A with Φ as program such that, for all $i \geq 1$, the mgu used during the i-th resolution step of the SLD-proof, is a renaming substitution whose domain coincides with the variables occurring in the head of the rule (i.e., the clause) used.*

Proof. If $A \in \mathsf{CoInd}(\Phi)$, then, by lemma 1, A is root of a proof tree T for Φ. Number the arcs emanating from each node from left to right, starting with 1. Each node can be designated (indexed) by the word obtained by concatenating the numbers of the arcs of the path leading from the root to the node (ε is the empty word). The breadth-first traversal of T produces a list $\mathcal{L}$. Since T is a proof tree for Φ, for each node $A_{\bar{\imath}}$ in T, there exists a clause $C_{T,\bar{\imath}} \in \Phi$ which can be written $A_{\bar{\imath}} \leftarrow A_{\bar{\imath}1}, \cdots, A_{\bar{\imath}n_{\bar{\imath}}}$. Indexes of T can be ordered as follows: $\bar{\imath} \prec \bar{\jmath}$ iff $A_{\bar{\imath}}$ occurs before $A_{\bar{\jmath}}$ in $\mathcal{L}$. $Z_T = \cup var(C_{T,\bar{\imath}})$ is the set, possibly infinite, of variables occurring in T. It can be proved [12] that given a clause $C_{T,\bar{\imath}} \in \Phi$, a renaming substitution $r_0^{\bar{\imath}}$, such that $range(r_0^{\bar{\imath}}) \cap var(C_{T,\bar{\imath}}) = \emptyset$, and a set of variables $Z_{\bar{\imath}}$,

there exists a substitution $\theta_{\bar{\imath}}$, a clause $C_{\bar{\imath}}$ and a renaming substitution $r_1^{\bar{\imath}} = r^{\bar{\imath}} r_0^{\bar{\imath}}$ such that:

$$var(C_{\bar{\imath}}) \cap (var(r_0^{\bar{\imath}} C_{T,\bar{\imath}}) \cup Z_{\bar{\imath}}) = \emptyset \qquad r_1^{\bar{\imath}} C^-_{T,\bar{\imath}} = \theta_{\bar{\imath}} C^-_{\bar{\imath}}$$
$$dom(\theta_{\bar{\imath}}) = var(C^+_{\bar{\imath}}) \qquad range(r^{\bar{\imath}}) = var(C^-_{\bar{\imath}}) \backslash var(C^+_{\bar{\imath}})$$

where $\theta_{\bar{\imath}}$ is an idempotent renaming substitution which is a mgu of $C^+_{\bar{\imath}}$ and $r_0^{\bar{\imath}} C^+_{T,\bar{\imath}}$. From $\mathcal{L}$, we can define the following sequence of resolution steps:

$$t_\varepsilon = \mathcal{T}(C_{T,\varepsilon}, s_{id}, Z_T),\ t_{\bar{\imath}k} = \mathcal{T}\left(C_{T,\bar{\imath}k}, r_0^{\bar{\imath}k}, Z_{\bar{\imath}k}\right) \begin{cases} 1 \leq k \leq n_{\bar{\imath}} \\ r_0^{\bar{\imath}k} = r_1^{\bar{\imath}} \\ Z_{\bar{\imath}k} = Z_T \cup \bigcup\limits_{\bar{\jmath} \prec \bar{\imath}k} var(C_{\bar{\jmath}}) \end{cases}$$

where $\mathcal{T}(C_{T,\bar{\imath}}, r_0^{\bar{\imath}}, Z_{\bar{\imath}})$ denotes the transition $r_0^{\bar{\imath}} C^+_{T,\bar{\imath}} \overset{C_{\bar{\imath}}, \theta_{\bar{\imath}}, Z_{\bar{\imath}}}{\rightarrow_\Phi} \theta_{\bar{\imath}} C^-_{\bar{\imath}}$ and where s_{id} is the empty substitution. This definition is sound since we can prove by induction that $\forall \bar{\imath}$, $range(r_0^{\bar{\imath}}) \cap var(C_{T,\bar{\imath}}) = \emptyset$. Furthermore, it can be proved [12] that this sequence defines an SLD-proof satisfying the desired properties (fairness follows from the breadth-first traversal of T).

In this section, proof trees for a rule set Φ have been related to SLD-proofs with Φ viewed as a program. The SLD-proofs obtained are such that the clauses used are not instantiated (they are just renamed). We will see that the appropriate rule set allowing to study infinite derivations, which do not compute infinite terms, is the rule set obtained from a program P by considering all the (finite) instances, not necessarily ground, of clauses in P. This corresponds to the $\mathcal{C}$-semantics approach [6].

3.2 SLD-Proofs over the Domain of Finite Terms

SLD-proofs over the domain of finite terms are SLD-derivations which do not compute infinite terms. In a more formal way, they can be defined by:

Definition 3. *An **SLD-proof over the domain of finite terms** is either a refutation or a fair infinite derivation:*

$$R_0 \overset{C_1, \theta_1}{\rightarrow_P} R_1 \rightarrow_P \cdots \rightarrow_P R_{i-1} \overset{C_i, \theta_i}{\rightarrow_P} R_i \rightarrow_P \cdots$$

such that $\forall k \geq 0\ \exists p > k\ \forall q \geq p \quad \theta_q \cdots \theta_p \cdots \theta_{k+1} R_k \approx \theta_p \cdots \theta_{k+1} R_k$.

It is important to note that it does not suffice that the condition holds for the initial query. For example, with $P = \{q \leftarrow p(x); p(f(x)) \leftarrow p(x)\}$, even if during the derivation starting from q, each mgu θ_i used is such that $\theta_i q = q$, this derivation computes the infinite term f^ω. A different characterisation of SLD-proofs over the domain of finite terms can be obtained from the following lemma, proved in [12], and used during the proof of lemma 4.

Lemma 3. *A derivation* $R_0 \overset{C_1,\theta_1}{\rightarrow_P} R_1 \rightarrow_P \cdots \rightarrow_P R_{i-1} \overset{C_i,\theta_i}{\rightarrow_P} R_i \rightarrow_P \cdots$ *is an infinite SLD-proof over the domain of finite terms iff* $\forall i \geq 0$, $\exists R$, $\forall n \geq i+1$, $\theta_n\theta_{n-1}\cdots\theta_{i+1}R_i \leq R$, *where* R *is a query (i.e.,* R *does not contain infinite atoms).*

$\mathcal{C}$-semantics results correspond to SLD-refutations :

$$S_P^{\mathcal{C}} = \{A \text{ (finite atom)} \mid A \overset{*,\theta}{\rightarrow} \square \text{ and } \theta A = A\} = \mathsf{lfp}(T_{\lceil P \rceil}) = \mathsf{Ind}(\lceil P \rceil)$$

Let us investigate infinite SLD-proofs over the domain of finite terms. The soundness theorem can be proved directly by using proof trees.

Theorem 1 (Soundness). *If there exists an SLD-proof over the domain of finite terms* $A_1, \cdots, A_n \overset{C_1,\theta_1}{\rightarrow_P} R_1 \rightarrow_P \cdots \rightarrow_P R_{i-1} \overset{C_i,\theta_i}{\rightarrow_P} R_i \rightarrow_P \cdots$, *then there exists* $k \geq 0$, *such that for all* i $(1 \leq i \leq n)$ $\theta_k \cdots \theta_1 A_i \in \mathsf{gfp}(T_{\lceil P \rceil})$.

Proof. We first prove the theorem for $n = 1$. By Tarski's theorem and by lemma 1, it suffices to prove that for a natural k, there exists a proof tree of $\theta_k \cdots \theta_1 A_1$ for $\lceil P \rceil$. For this, let us define the sequence $T_1, \cdots, T_i, \cdots$ of partial proof trees, such that every atom occurring in R_i is a leaf in T_i, which is a partial proof tree of $\theta_i \cdots \theta_1 A_1$ for $\lceil P \rceil$. T_1 is obtained from the first transition: its root is $\theta_1 A_1$ whose children (which are leaves) are all the atoms occurring in $\theta_1 C_1^-$. Since $\theta_1 C_1 \in \lceil P \rceil$ and $\theta_1 A_1 = \theta_1 C_1^+$, T_1 is a partial proof tree of $\theta_1 A_1$ for $\lceil P \rceil$. Suppose now that T_{n-1} is a partial proof tree of $\theta_{n-1} \cdots \theta_1 A_1$ for $\lceil P \rceil$ (corresponding to the $n-1$ first transitions) such that atoms in R_{n-1} are leaves of T_{n-1}. By applying the substitution θ_n to each node of T_{n-1}, we get a partial proof tree of $\theta_n \cdots \theta_1 A_1$ for $\lceil P \rceil$ such that atoms in $\theta_n R_{n-1}$ are leaves. If A is the selected atom in R_{n-1}, then A is a leaf of T_{n-1} and $\theta_n A$ is a leaf in the new partial proof tree. Now, it suffices to add all the atoms in $\theta_n C_n^-$ as children of $\theta_n A$ (these children are leaves). In this way, we obtain a partial proof tree T_n satisfying the desired properties. Because the derivation does not compute infinite terms and therefore there exists a natural $k \geq 0$ such that for all $q \geq k$, $\theta_q \cdots \theta_k \cdots \theta_1 A_1 \approx \theta_k \cdots \theta_1 A_1$, by iterating this process, we obtain a proof tree of $\theta_k \cdots \theta_1 A_1$ for $\lceil P \rceil$. Furthermore each leaf corresponds to a unit clause of $\lceil P \rceil$ since the derivation is fair. For $n > 1$, the proof is similar: instead of building a sequence of partial proof trees, we build a sequence $((T_1^1, \cdots, T_1^n), \cdots, (T_i^1, \cdots, T_i^n), \cdots)$ of tuples of n partial proof trees for $\lceil P \rceil$ such that $\theta_i \cdots \theta_1 A_j$ is the root of T_i^j $(1 \leq j \leq n)$ and each atom occurring in R_i is a leaf of T_i^j for a j.

Since, by lemma 2, there exists an SLD-proof with the program $\lceil P \rceil$ from each atom occurring in $\mathsf{CoInd}(\lceil P \rceil)$, lemma 4 describes how to "translate" an SLD-derivation with $\lceil P \rceil$ into an SLD-derivation with P. It can be viewed as a "program lifting lemma" playing the same role as the (classical) lifting lemma in the proof of the (classical) completeness theorem.

Lemma 4 (Program lifting lemma). *If there exists an SLD-proof:* $A_0 \overset{C_1,\theta_1}{\rightarrow_{\lceil P \rceil}} R_1 \rightarrow_{\lceil P \rceil} \cdots \rightarrow_{\lceil P \rceil} R_{i-1} \overset{C_i,\theta_i}{\rightarrow_{\lceil P \rceil}} R_i \rightarrow_{\lceil P \rceil} \cdots$ *such that for all* $i \geq 1$, θ_i *is*

an idempotent renaming substitution such that $dom(\theta_i) = var(C_i^+)$*, then there exists an SLD-proof over a the domain of finite terms:* $A_0 \overset{C_{P,1},\sigma_1}{\rightarrow_P} R'_1 \rightarrow_P \cdots \rightarrow_P R'_{i-1} \overset{C_{P,i},\sigma_i}{\rightarrow_P} R'_i \rightarrow_P \cdots$ *such that for all* $i \geq 1$, $\sigma_i A_0 = A_0$, $C_i = \mu_i C_{P,i}$ *and* $R_i = \rho_i R'_i$ *where* ρ_i *is the restriction of* $\theta_i\mu_i\theta_{i-1}\mu_{i-1}\cdots\theta_1\mu_1$ *to the variables occurring in* R'_i*.*

Proof. It can be proved [12] that there exists a set $\{C_{P,1}, \cdots, C_{P,i}, \cdots\}$ of variants of clauses of P such that each $C_{P,i}$ satisfies $C_i = \mu_i C_{P,i}$ where μ_i is an idempotent substitution such that $dom(\mu_i) = var(C_{P,i})$ and

$$\forall i > 0 \quad var(C_{P,i}) \cap (var(A_0) \cup \cup_{1 \leq j < i} var(C_{P,j}) \cup \cup_{j \geq 1} var(C_j)) = \emptyset$$

• *For the first transition.* By definition $\theta_1 A_0 = \theta_1 C_1^+ = \theta_1\mu_1 C_{P,1}^+$. Furthermore, since $var(C_{P,1}) \cap var(A_0) = \emptyset$, and $var(C_1) \cap var(A_0) = \emptyset$, we have $\theta_1\mu_1 A_0 = A_0 = \theta_1\mu_1 C_{P,1}^+$. It can be proved [12] that the restriction σ_1 of $\theta_1\mu_1$ to $var(C_{P,1}^+)$ is a mgu of A_0 and $C_{P,1}^+$ and we get the transition $A_0 \overset{C_{P,1},\sigma_1}{\rightarrow_P} R'_1$. $\sigma_1 A_0 = A_0$ follows from $\theta_1\mu_1 A_0 = A_0$. Now, let ρ_1 be the restriction of $\theta_1\mu_1$ to $var(R'_1)$ and let us prove that $\rho_1 R'_1 = \rho_1\sigma_1 C_{P,1}^- = \theta_1\mu_1 C_{P,1}^- = \theta_1 C_1^- = R_1$. Let $v \in var(C_{P,1}^-)$, two cases are possible. If $v \in var(C_{P,1}^+)$, then $\sigma_1 v = \theta_1\mu_1 v$ and we can conclude since $\theta_1\mu_1\theta_1\mu_1 v = \theta_1\mu_1 v$. Else, if $v \notin var(C_{P,1}^+)$, then we have $\rho_1\sigma_1 v = \rho_1 v = \theta_1\mu_1 v$ which settles the claim.
• *For the i-th transition.* If A is the selected atom in R_{i-1} at position k, then there exists an atom A' occurring at position k in R'_{i-1} such that $A = \rho_{i-1}A'$ and we get $\theta_i\rho_{i-1}A' = \theta_i\mu_i C_{P,i}^+$. From $dom(\rho_{i-1}) \subseteq var(R'_{i-1})$ and $var(R'_{i-1}) \subseteq (\cup_{1 \leq j < i} var(C_{P,j}) \cup var(A_0))$, it follows $\rho_{i-1}C_{P,i} = C_{P,i}$ and therefore $\theta_i\rho_{i-1}A' = \theta_i\mu_i\rho_{i-1}C_{P,i}^+$. Furthermore, $dom(\mu_i) = var(C_{P,i})$ and we have $\mu_i A = A$. Hence we have $\theta_i\mu_i\rho_{i-1}A' = \theta_i\mu_i\rho_{i-1}C_{P,i}^+$, and since $\theta_i\mu_i\cdots\theta_1\mu_1 A_0 = A_0$, there exists a mgu σ_i of A' and $C_{P,i}^+$ such that $\sigma_i A_0 = A_0$ and for a substitution η_i, we have $\eta_i\sigma_i = \theta_i\mu_i\rho_{i-1}$. Hence, we get the transition $R'_{i-1} \overset{C_{P,i},\sigma_i}{\rightarrow_P} R'_i$. Since σ_i is idempotent, we have $\theta_i\mu_i\rho_{i-1}\sigma_i = \eta_i\sigma_i\sigma_i = \eta_i\sigma_i = \theta_i\mu_i\rho_{i-1}$ and it follows $\theta_i\mu_i\rho_{i-1}R'_i = R_i$. Finally, we prove by induction that $\forall n \geq i+1$, $\rho_n\sigma_n\sigma_{n-1}\cdots\sigma_{i+1}R'_i = R_i$. Hence, since R_i contains only finite atoms and for all $n \geq i+1$, we have $\sigma_n\sigma_{n-1}\cdots\sigma_{i+1}R'_i \leq R_i$, by lemma 3, the derivation obtained is a derivation over the domain of finite terms.

We are now in position to prove the completeness theorem.

Theorem 2 (Completeness). *If* $A \in \mathsf{gfp}(T_{\lceil P \rceil})$*, then there exists an SLD-proof over the domain of finite terms:*

$$A \overset{C_{P,1},\sigma_1}{\rightarrow_P} R'_1 \rightarrow_P \cdots \rightarrow_P R'_{i-1} \overset{C_{P,i},\sigma_i}{\rightarrow_P} R'_i \rightarrow_P \cdots$$

such that for all $i \geq 1$, $\sigma_i A = A$

Proof. Immediate by Tarski's theorem, lemma 2 and lemma 4.

Hence, if we consider program (1), since $\mathsf{eat}(i) \in \mathsf{gfp}(T_{\lceil P \rceil})$, there exists an SLD-proof over the domain of finite terms from $\mathsf{eat}(i)$. Another typical example is the program containing the clauses :

$$\mathsf{path}(x,x) \leftarrow \quad ; \quad \mathsf{path}(x,z) \leftarrow \mathsf{edge}(x,y), \mathsf{path}(y,z)$$

and testing connectivity in a directed graph. When the graph considered is cyclic, there exists an infinite derivation over the domain of finite terms from the query $\mathsf{path}(s,x)$, where s is an arbitrary node occurring in the cycle, which can be viewed as a proof of $\forall x\, \mathsf{path}(s,x)$.

3.3 Infinite SLD-Derivations Which Do Not Compute Anything

The derivation obtained by lemma 2 is a special case of a derivation which does not compute infinite terms: such a derivation does not compute anything since the mgu's used are just renaming substitutions. For this subclass of derivations, we can prove a supplementary result concerning unfair derivations. For this, let $\lceil P \rceil^+ = \{\theta C \mid C \in P\,, dom(\theta) \subseteq var(C^+)\}$ and from which we can define a monotone operator $T_{\lceil P \rceil^+}$, as described by (4). Unfair infinite derivations which do not compute anything can be viewed as partial proofs. Recall that given a derivation:

$$R_0 \overset{C_1,\theta_1}{\rightarrow}_P R_1 \rightarrow_P \cdots \rightarrow_P R_{i-1} \overset{C_i,\theta_i}{\rightarrow}_P R_i \rightarrow_P \cdots$$

for all $i \geq 1$ we have $P \models R_i \Rightarrow P \models \theta_i \cdots \theta_1 R_0$. This result can be generalised by considering $R_\infty = \cup_{p \geq 0} \cap_{p \leq n} R_n$.

Theorem 3. *Let P be a program and A_0 be an atom. If there exists an infinite derivation $R_0 = A_0 \overset{C_1,\theta_1}{\rightarrow}_P R_1 \rightarrow_P \cdots \rightarrow_P R_{i-1} \overset{C_i,\theta_i}{\rightarrow}_P R_i \rightarrow_P \cdots$ such that for all $i > 0$, $dom(\theta_i) \subseteq var(C_i^+)$, then:*

$$R_\infty \subseteq \mathsf{gfp}(T_{\lceil P \rceil^+}) \Rightarrow A_0 \in \mathsf{gfp}(T_{\lceil P \rceil^+})$$

Proof. Suppose that $\cup_{p \geq 0} \cap_{p \leq n} R_n \subseteq \mathsf{gfp}(T_{\lceil P \rceil^+})$ and let us prove that $A_0 \in \mathsf{gfp}(T_{\lceil P \rceil^+})$. For this, by Tarski's theorem and by lemma 1, it suffices to prove that there exists a proof tree T of A_0 for $\lceil P \rceil^+$. Let us define the sequence $T_1, \cdots, T_i, \cdots$ of partial proof trees such that every atom occurring in R_i is a leaf in T_i. T_1 is obtained by considering the first transition: its root $A_0 = \theta_1 A_0$ has atoms in $R_1 = \theta_1 C_1^-$ as children. We show now how we can obtain T_n from T_{n-1}. We know that atoms occurring in R_{n-1} are leaves in T_{n-1}. Let A be the selected atom in R_{n-1}, since $dom(\theta_n) \subseteq var(C_n^+)$, T_n is obtained by adding atoms occurring in $\theta_n C_n^-$ as children of A. Since, $R_n = \theta_n R_{n-1}[k \leftarrow C_n^-] = R_{n-1}[k \leftarrow \theta_n C_n^-]$, T_n is a partial proof tree of A_0 for $\lceil P \rceil^+$ such that every atom occurring in R_n is a leaf in T_n. By iterating this process, we obtain a partial proof tree T_∞ whose leaves are either the head of a unit clause in $\lceil P \rceil^+$ or an atom in R_∞, which is, by hypothesis, in $\mathsf{gfp}(T_{\lceil P \rceil^+})$ and correspond, by Tarski's theorem and by lemma 1, to the root of a proof tree for $\lceil P \rceil^+$. Therefore, by adding in T_∞ these proof trees at the corresponding leaf, we obtain a proof tree of A_0 for $\lceil P \rceil^+$.

This theorem is not a special case of theorem 2, it just gives another way to interpret a subclass of infinite derivations. For example, if we consider the derivation we can obtain with program (2), from the query $p(x)$, then, by theorem 2, we have $p(x) \in \mathsf{gfp}(T_{\lceil P \rceil})$ while by theorem 3, we just have $p(x) \in \mathsf{gfp}(T_{\lceil P \rceil^+}) \Rightarrow p(x) \in \mathsf{gfp}(T_{\lceil P \rceil^+})$ since for this derivation we have $R_\infty = p(x)$. Such a semantics works well for programs whose clauses do not contain existential variables (i.e., $var(C^-) \subseteq var(C^+)$), since in this case we have $\lceil P \rceil = \lceil P \rceil^+$.

4 Conclusion

In this paper, semantics of nonterminating derivations has been investigated within a proof-theoretic framework: clauses have been considered as constructors of a co-inductive definition. Following this approach, a semantics for the class of infinite derivations which do not compute infinite terms has been defined and proved sound and complete by using purely proof-theoretic methods: an atom is the starting point of an infinite derivation over the domain of finite terms if and only if it is in the greatest fixpoint of the transformation $T_{\lceil P \rceil}$. The restriction to the class of derivations over the domain of finite terms is justified by incompleteness results of other approaches, considering infinite terms, in which the greatest fixpoint construction, corresponding to the "logic programs as co-inductive definitions" paradigm, is not equivalent to the operational semantics. In fact, in section 2, by considering this identification at a deeper level, we have seen that co-induction is too rich to give a semantics to nonterminating SLD-derivations. This observation explains why most attempts to give a complete semantics to derivations computing infinite terms have not been successful. Therefore, while all the approaches existing in this area are based on the concept of "atoms computable at infinity", we have presented a semantics based on the concept of "atoms provable at infinity".

It seems that the operational notion of "computability at infinity" (associated with infinite derivations computing infinite terms) is better captured by a least fixpoint characterisation. This idea has been developped by G. Levi and C. Palamidessi in [15]. In an order-theoretic framework (involving algebraic CPO), they consider the "final result" of an infinite derivation as the limit of a sequence of approximations, characterised by a least fixpoint semantics based on a modified version of the programs. Then, infinite objects in the denotation of a program are characterised by the topological closure of $\mathsf{lfp}(T_{P \cup C(P)})$ (where $C(P)$ is the set of added clauses): each infinite element is the least upper bound of a directed set (of finite elements which are its partial approximations) included in $\mathsf{lfp}(T_{P \cup C(P)})$. However, the semantics obtained is sound but not complete.

Of course, a satisfactory semantics for all infinite derivations from a logic program has not yet been found. However, even if the results proved in this paper may seem unsurprising, they allow us to gain a better understanding of the problem. In fact, current approaches in this area only give meaning to infinite derivations that compute at least an infinite term and ignore derivations over the domain of finite terms.

Acknowledgements. Many thanks to René Lalement and Catherine Dubois for enlightening discussions about this work.

References

1. M.A. Nait Abdallah. On the interpretation of infinite computations in logic programming. In J. Paredaens, editor, *11th International Colloquium on Automata, Languages and Programming, ICALP'84*, volume 172 of *LNCS*, pages 358–370. Springer-Verlag, 1984.
2. P. Aczel. An introduction to inductive definitions. In K.J. Barwise, editor, *Handbook of Mathematical Logic*, Studies in Logic and Foundations of Mathematics. North Holland, 1977.
3. T. Coquand. Infinite objects in type theory. In H. Barendregt and T. Nipkow, editors, *1st International Workshop on Types for Proofs and Programs, TYPES'93*, volume 806 of *LNCS*, pages 62–78. Springer-Verlag, 1994.
4. F.S. de Boer, A. Di Pierro, and C. Palamidessi. Nondeterminism and infinite computations in constraint programming. *Theoretical Computer Science*, 151(1):37–78, 1995.
5. P. Deransart and J. Maluszynski. *A Grammatical View of Logic Programming.* The MIT Press, 1993.
6. M. Falaschi, G. Levi, C. Palamidessi, and M. Martelli. Declarative modeling of the operational behavior of logic languages. *Theoretical Computer Science*, 69(3):289–318, 1989.
7. W.G. Golson. Toward a declarative semantics for infinite objects in logic programming. *Journal of Logic Programming*, 5(2):151–164, 1988.
8. J. Hein. Completions of perpetual logic programs. *Theoretical Computer Science*, 99(1):65–78, 1992.
9. G. Huet. *A Uniform Approach to Type Theory*, volume Logical Foundations of Functional Programming, pages 337–398. Addison-Wesley, 1990.
10. J. Jaffar and J.L. Lassez. Constraint Logic Programming. Technical Report 86/74, Monash University, Victoria, Australia, June 1986.
11. J. Jaffar and P.J. Stuckey. Semantics of infinite tree logic programming. *Theoretical Computer Science*, 46(2-3):141–158, 1986.
12. M. Jaume. Logic programming and co-inductive definitions. Research Report 98-140, Enpc-Cermics, 1998.
13. M. Jaume. A full formalisation of SLD-resolution in the calculus of inductive constructions. *Journal of Automated Reasoning*, 23(3–4):347–371, 1999.
14. R. Lalement. *Computation as Logic.* Prentice Hall International Series in Computer Science, 1993.
15. G. Levi and C. Palamidessi. Contributions to the semantics of logic perpetual processes. *Acta Informatica*, 25(6):691–711, 1988.
16. J.W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, 1987.
17. L.C. Paulson and A.W. Smith. Logic programming, functional programming, and inductive definitions. In P. Schroeder-Heister, editor, *International Workshop on Extensions of Logic Programming*, volume 475 of *LNCS*, pages 283–310. Springer-Verlag, 1989.
18. A. Podelski, W. Charatonik, and M. Müller. Set-based failure analysis for logic programs and concurrent constraint programs. In S. Doaitse Swierstra, editor, *8th European Symposium on Programming, ESOP'99*, LNCS, pages 177–192. Springer-Verlag, 1999.

A Theory of Explicit Mathematics Equivalent to ID_1

Reinhard Kahle[1] and Thomas Studer[2]

[1] WSI, Universität Tübingen,
Sand 13, D-72076 Tübingen, Germany
Tel. +49-7071-29 74036, Fax: +49-7071-29 5060
kahle@informatik.uni-tuebingen.de
[2] IAM, Universität Bern,
Neubrückstr. 10, CH-3012 Bern, Switzerland
Tel. +41-31-631 4976, Fax: +41-31-631 3965
tstuder@iam.unibe.ch

Abstract. We show that the addition of *name induction* to the theory $\mathsf{EETJ} + (\mathcal{L}_{\mathsf{EM}}\text{-}\mathsf{I}_{\mathsf{N}})$ of explicit elementary types with join yields a theory proof-theoretically equivalent to ID_1.

Keywords: Proof theory, explicit mathematics, inductive definitions.

1 Introduction

In this paper, we introduce a theory of explicit mathematics which is proof-theoretically equivalent to the well-known theory ID_1 of non-iterated positive arithmetical inductive definitions.

Explicit mathematics was introduced by Feferman to formalize Bishop-style constructive mathematics [Fef75,Fef79]. In the following, it turned out that this framework is important for proof-theoretic studies of subsystems of analysis and Kripke-Platek set theory. Moreover, it provides a very useful account to theoretical computer science, particularly, it is well-suited for the study of functional and object-oriented programming, cf. [Fef90,Fef91,Fef92,Stä97,Stä98,Stu0x].

Theories of explicit mathematics are formulated in a two sorted language. The first-order part, consisting of so-called *applicative theories*, is based on partial combinatory logic which can be extended axiomatically by additional constants, cf. [JKS99]. *Types* build the second sort of objects in explicit mathematics. They are extensional in the usual set-theoretic sense, but a special naming relation due to Jäger [Jäg88] allows us to deal with *names* of the types on the first-order level. These names show an intensional behaviour.

There exist a wide variety of theories of explicit mathematics. The proof-theoretic strength of the different theories cover a broad part of the landscape of mathematical theories. Nevertheless, the theory presented here is the first theory of explicit mathematics equivalent to ID_1.

The well-known theory ID_1 of non-iterated inductive definitions is one of the most prominent theories in proof theory. Formalizing least fixed points of

P. Clote and H. Schwichtenberg (Eds.): CSL 2000, LNCS 1862, pp. 356–370, 2000.

positive arithmetical operator forms, it can be regarded as the most elementary *impredicative* theory. Going back to Kreisel [Kre63], its proof-theoretic study (and the study of its iterations) can be found in [Fef70,BFPS81,Poh89].

In order to get a theory with the proof-theoretic strength of ID_1, we will add the concept of *name induction* to the theory EETJ of *explicit elementary types with join*. That means that names of types can be built by use of *generators* only, i.e. that the naming relation $\Re$ is, so to say, *least*.

In the context of Martin-Löf's type theory, this leastness condition corresponds to certain elimination rules which have first been considered by Palmgren and later by Rathjen, also in connection with *universes*, [Pal98,GR94]. For applicative theories, the concept of name induction in the presence of universes is studied in detail in a joint work with Jäger, [JKS0x]. The theories studied in that paper exceed the strength of ID_1 substantially by having proof-theoretic strength of Feferman's theory T_0. For the notion of proof-theoretic strength, we refer to Feferman [Fef88,Fef0x].

In type systems dealing with record or object types the concept of structural rule is important. Simplifying, we can say that these rules rely on the assumption that the universe of types consists of record or object types only, cf. e.g. [AC96]. Name induction can be seen as a generalization of this idea since it allows us to prove that the only types that exists are those which are created by the generators.

The structure of the paper is as follows. In the next section, we introduce the theory NEM of explicit mathematics with name induction and state some basic results. As the core of the paper, we prove in Section 3 that NEM allows for the definition of *accessible parts*. This result is used in the fourth section to give an interpretation of $\mathsf{ID}_1^{\mathsf{acc}}$, a theory equivalent to ID_1, in NEM. In the final section, we describe a model of NEM which can be formalized in ID_1.

A substantial part of the work of the first author was elaborated while visiting Sol Feferman at Stanford University under support of the Deutsche Forschungsgemeinschaft. The work of the second author is supported by the Schweizerische Nationalfonds. This article benefits from fruitful discussion with Gerhard Jäger.

2 The Theory NEM of Explicit Mathematics with Name Induction

2.1 Explicit Mathematics

In this section, we present the theory EETJ of explicit elementary types with join.

The underlying language $\mathcal{L}_{\mathsf{EM}}$ is comprised of

- individual variables $a, b, c, f, u, v, w, x, y, z, \ldots$,
- type variables $A, B, S, T, U, V, X, Y, Z, \ldots$,
- individual constants k, s (combinators), $\mathsf{p}, \mathsf{p}_0, \mathsf{p}_1$ (pairing and projections), 0 (zero), s_{N} (successor), p_{N} (predecessor) and d_{N} (definition by numerical cases),

- *generators* which are special individual constants, namely nat (natural numbers), id (identity), co (complement), int (intersection), dom (domain), inv (inverse image) and j (join),
- one binary function symbol $\cdot$ for (partial) application of individuals to individuals,
- unary relation symbols $\downarrow$ (defined) and N (natural numbers) and
- binary relation symbols $\in$ (membership), $=$ (equality) and $\Re$ (naming or representation).

Individual terms $(r, s, t, r_1, s_1, t_1, \ldots)$ of $\mathcal{L}_{\mathsf{EM}}$ are built up from individual variables and individual constants by means of the function symbol $\cdot$. We use (st) or st as an abbreviation for $(s \cdot t)$ and adopt the convention of association to the left, i.e. $s_1 s_2 \ldots s_n$ stands for $(\ldots(s_1 \cdot s_2)\ldots s_n)$.

Atomic formulae of $\mathcal{L}_{\mathsf{EM}}$ are $\mathsf{N}(s)$, $s\downarrow$, $s = t$, $U = V$, $s \in U$ and $\Re(s, U)$. $\mathsf{N}(s)$ means that s is a natural number. $s\downarrow$ means that s *is defined* or s *has a value*. $\Re(s, U)$ is the naming relation, expressing that the individual s *represents* the type U or is a *name* of U.

The *formulae* of $\mathcal{L}_{\mathsf{EM}}$ $(\varphi, \psi, \ldots)$ are built up from the atomic formulae by use of the usual propositional connectives and quantification in both sorts, over individuals as well as over types.

A formula which contains neither quantifiers over types nor the naming relation $\Re$ is called *elementary*.

As abbreviations, we use:

$$
\begin{aligned}
t' &:= \mathsf{s_N} t,\\
(s, t) &:= \mathsf{p} st,\\
s \simeq t &:= s\downarrow \vee t\downarrow \to s = t,\\
s \neq t &:= s\downarrow \wedge t\downarrow \wedge \neg(s = t),\\
s \in \mathsf{N} &:= \mathsf{N}(s),\\
\exists x \in \mathsf{N}.\varphi(x) &:= \exists x.x \in \mathsf{N} \wedge \varphi(x),\\
\forall x \in \mathsf{N}.\varphi(x) &:= \forall x.x \in \mathsf{N} \to \varphi(x),\\
s \mathrel{\dot\in} t &:= \exists X.\Re(t, X) \wedge s \in X,\\
\exists x \mathrel{\dot\in} s.\varphi(x) &:= \exists x.x \mathrel{\dot\in} s \wedge \varphi(x),\\
\forall x \mathrel{\dot\in} s.\varphi(x) &:= \forall x.x \mathrel{\dot\in} s \to \varphi(x),\\
\Re(s) &:= \exists X.\Re(s, X).
\end{aligned}
$$

The logic for the first-order part of theories of explicit mathematics is Beeson's classical *logic of partial terms*, cf. [Bee85,TvD88]. The second order part is based on classical logic with equality.

The nonlogical axioms of EETJ can be divided into the following groups.

I. Applicative axioms.

(1) $\mathsf{k}ab = a$,
(2) $\mathsf{s}ab{\downarrow} \wedge \mathsf{s}abc \simeq ac(bc)$,
(3) $\mathsf{p}_0(a,b) = a \wedge \mathsf{p}_1(a,b) = b$,
(4) $0 \in \mathsf{N} \wedge \forall x \in \mathsf{N}.x' \in \mathsf{N}$,
(5) $\forall x \in \mathsf{N}.x' \neq 0 \wedge \mathsf{p}_\mathsf{N}(x') = x$,
(6) $\forall x \in \mathsf{N}.x \neq 0 \rightarrow \mathsf{p}_\mathsf{N}x \in \mathsf{N} \wedge (\mathsf{p}_\mathsf{N}x)' = x$,
(7) $a \in \mathsf{N} \wedge b \in \mathsf{N} \wedge a = b \rightarrow \mathsf{d}_\mathsf{N}xyab = x$,
(8) $a \in \mathsf{N} \wedge b \in \mathsf{N} \wedge a \neq b \rightarrow \mathsf{d}_\mathsf{N}xyab = y$.

II. Explicit representation and extensionality.

(1) $\exists x.\Re(x,U)$,
(2) $\Re(a,U) \wedge \Re(a,V) \rightarrow U = V$,
(3) $(\forall x.x \in U \leftrightarrow x \in U) \rightarrow U = V$.

III. Basic type existence axioms.

Natural numbers

$\Re(\mathsf{nat}) \wedge \forall x.x \mathrel{\dot\in} \mathsf{nat} \leftrightarrow \mathsf{N}(x)$.

Identity

$\Re(\mathsf{id}) \wedge \forall x.x \mathrel{\dot\in} \mathsf{id} \leftrightarrow \exists y.x = (y,y)$.

Complements

$\Re(a) \rightarrow \Re(\mathsf{co}(a)) \wedge \forall x.x \mathrel{\dot\in} \mathsf{co}(a) \leftrightarrow x \mathrel{\dot\notin} a$.

Intersections

$\Re(a) \wedge \Re(b) \rightarrow \Re(\mathsf{int}(a,b)) \wedge \forall x.x \mathrel{\dot\in} \mathsf{int}(a,b) \leftrightarrow x \mathrel{\dot\in} a \wedge x \mathrel{\dot\in} b$.

Domains

$\Re(a) \rightarrow \Re(\mathsf{dom}(a)) \wedge \forall x.x \mathrel{\dot\in} \mathsf{dom}(a) \leftrightarrow \exists y.(x,y) \mathrel{\dot\in} a$.

Inverse images

$\Re(a) \rightarrow \Re(\mathsf{inv}(a,f)) \wedge \forall x.x \mathrel{\dot\in} \mathsf{inv}(a,f) \leftrightarrow fx \mathrel{\dot\in} a$.

Joins

$\Re(a) \wedge (\forall x \mathrel{\dot\in} a.\Re(fx)) \rightarrow \Re(\mathsf{j}(a,f)) \wedge \Sigma(a,f,\mathsf{j}(a,f))$,

where $\Sigma(a,f,b)$ means that b names the disjoint union of f over a, defined as

$$\Sigma(a,f,b) := \forall x.x \mathrel{\dot\in} b \leftrightarrow \exists y,z.x = (y,z) \wedge y \mathrel{\dot\in} a \wedge z \mathrel{\dot\in} fy.$$

IV. Uniqueness of generators. With respect to $\mathcal{L}_\mathsf{EM}$, it is given by the collection ($\mathcal{L}_\mathsf{EM}$-UG) of the following axioms for all syntactically different generators r_0 and r_1 and arbitrary generators s and t of $\mathcal{L}_\mathsf{EM}$:

(1) $r_0 \neq r_1$,
(2) $\forall x.sx \neq \mathsf{nat} \wedge sx \neq \mathsf{id}$,
(3) $\forall x, y.sx = ty \rightarrow s = t \wedge x = y$.

EETJ is the theory consisting of all axioms of the groups I. – IV.

As addition to the axioms of EETJ, we will consider the induction principle ($\mathcal{L}_{\mathsf{EM}}$-$\mathsf{I}_{\mathsf{N}}$), the schema of complete induction on N for arbitrary formulae $\varphi(u)$:

($\mathcal{L}_{\mathsf{EM}}$-$\mathsf{I}_{\mathsf{N}}$) $\qquad \varphi(0) \wedge (\forall x \in \mathsf{N}.\varphi(x) \rightarrow \varphi(x')) \rightarrow \forall x \in \mathsf{N}.\varphi(x)$

It is a well-known result that we can introduce λ abstraction and recursion using the combinator axioms (1) and (2), cf. [Fef75, Bee85].

Proposition 1.

1. *For every variable x and every term t of $\mathcal{L}_{\mathsf{EM}}$, there exists a term $\lambda x.t$ of $\mathcal{L}_{\mathsf{EM}}$ whose free variables are those of t, excluding x, such that*

$$\mathsf{EETJ} \vdash \lambda x.t \downarrow \wedge\, (\lambda x.t)\, x \simeq t.$$

2. *There exists a term* rec *of $\mathcal{L}_{\mathsf{EM}}$ such that*

$$\mathsf{EETJ} \vdash \mathsf{rec}\, f \downarrow \wedge\, \forall x.\mathsf{rec}\, f\, x \simeq f\, (\mathsf{rec}\, f)\, x.$$

Our definition EETJ is based on a finite axiomatization of elementary comprehension. This approach is essential for the formulation of name induction below. In contrast, the original definition of EETJ employed an infinite axiom schema. A theorem of Feferman and Jäger [FJ96] shows that this schema is derivable from the finite axiomatization.

Lemma 1 (Elementary comprehension). *Let φ be an elementary $\mathcal{L}_{\mathsf{EM}}$ formula with no (distinct) individual variables other than $z_1, \ldots, z_{m+1}$ and no (distinct) type variables other than $Z_1, \ldots, Z_n$. Then there exists a closed individual term t of $\mathcal{L}_{\mathsf{EM}}$, depending on φ, such that* EETJ *proves for all individual terms $\boldsymbol{a} = a_1, \ldots, a_m$, $\boldsymbol{b} = b_1, \ldots, b_n$ and type terms $\boldsymbol{S} = S_1, \ldots, S_n$ that:*

1. $\Re(\boldsymbol{b}, \boldsymbol{S}) \rightarrow \Re(t(\boldsymbol{a}, \boldsymbol{b}))$,
2. $\Re(\boldsymbol{b}, \boldsymbol{S}) \rightarrow \forall x(x \mathrel{\dot\in} t(\boldsymbol{a}, \boldsymbol{b}) \leftrightarrow \varphi[x, \boldsymbol{a}, \boldsymbol{S}])$.

Informally, we will write $\{x : \varphi(x)\}$ for the collection of all individuals c satisfying $\varphi(c)$. Using this notation, the lemma expresses that, for elementary formulae $\varphi[u, \boldsymbol{y}, \boldsymbol{Y}]$, the following hold:

1. $\{x : \varphi[x, \boldsymbol{a}, \boldsymbol{S}]\}$ is a type,
2. there is a name $t(\boldsymbol{a}, \boldsymbol{b})$ for this type which is given uniformly in the individual parameters and the names of the type parameters.

2.2 Name Induction

In this section, we define the schema of *name induction*. This induction principle states that names can be defined by means of generators only. Because, in a certain sense, names can be seen as intensional representations of sets, we get an intensional version of $\in$ induction.

In order to state the formal definition of name induction, we introduce as auxiliary notation the closure condition $\mathcal{C}(\varphi, a)$ as the disjunction of the following formulae:

(1) $a = \mathsf{nat} \vee a = \mathsf{id}$,
(2) $\exists x.a = \mathsf{co}(x) \wedge \varphi(x)$,
(3) $\exists x, y.a = \mathsf{int}(x, y) \wedge \varphi(x) \wedge \varphi(y)$,
(4) $\exists x.a = \mathsf{dom}(x) \wedge \varphi(x)$,
(5) $\exists f, x.a = \mathsf{inv}(f, x) \wedge \varphi(x)$,
(6) $\exists f, x.a = \mathsf{j}(x, f) \wedge \varphi(x) \wedge \forall y \mathrel{\dot\in} x.\varphi(fy)$.

The schema of name induction is now given by

$$(\mathcal{L}_{\mathsf{EM}}\text{-}\mathsf{I}_{\Re}) \qquad (\forall x.\mathcal{C}(\varphi, x) \to \varphi(x)) \to \forall x.\Re(x) \to \varphi(x),$$

for arbitrary formulae $\varphi(x)$ of $\mathcal{L}_{\mathsf{EM}}$.

The theory NEM of *explicit mathematics with name induction* consists of the axioms of EETJ plus $(\mathcal{L}_{\mathsf{EM}}\text{-}\mathsf{I}_{\mathsf{N}})$ and $(\mathcal{L}_{\mathsf{EM}}\text{-}\mathsf{I}_{\Re})$.

As a first consequence of $(\mathcal{L}_{\mathsf{EM}}\text{-}\mathsf{I}_{\Re})$, we prove *name strictness* which, more explicitly, says the (appropriate) arguments of generators of names are names, too. This is represented by the conjunction $\mathsf{Str}(\Re)$ of the following clauses:

(1) $\forall x.\Re(\mathsf{co}(x)) \to \Re(x)$,
(2) $\forall x, y.\Re(\mathsf{int}(x, y)) \to \Re(x) \wedge \Re(y)$,
(3) $\forall x.\Re(\mathsf{dom}(x)) \to \Re(x)$,
(4) $\forall f, x.\Re(\mathsf{inv}(f, x)) \to \Re(x)$,
(5) $\forall f, x.\Re(\mathsf{j}(x, f)) \to \Re(x) \wedge \forall y \mathrel{\dot\in} x.\Re(fy)$.

To show $\mathsf{Str}(\Re)$ in NEM, we first note that the closure of the names under condition $\mathcal{C}$ is guaranteed by the type existence axioms of EETJ:

$$\mathsf{EETJ} \vdash \mathcal{C}(\Re, x) \to \Re(x).$$

Lemma 2. $\mathsf{NEM} \vdash \mathsf{Str}(\Re)$.

Proof. The proof is straightforward using $(\mathcal{L}_{\mathsf{EM}}\text{-}\mathsf{I}_{\Re})$ on the formula $\mathcal{C}(\Re, x)$, i.e. we have

$$(\forall x.\mathcal{C}(\mathcal{C}(\Re, x), x) \to \mathcal{C}(\Re, x)) \to \forall x.\Re(x) \to \mathcal{C}(\Re, x).$$

The premise follows immediately from the preceding remark and the fact that φ occurs only positively in $\mathcal{C}(\varphi, x)$. From the consequence $\forall x.\Re(x) \to \mathcal{C}(\Re, x)$

we get the required conclusion $\mathsf{Str}(\Re)$ by substituting the different names. For example, for clause (5) we have

$$\begin{aligned}\Re(\mathsf{j}(x,f)) &\to \mathcal{C}(\Re,\mathsf{j}(x,f))\\ &\to \exists g,z.\mathsf{j}(x,f)=\mathsf{j}(z,g)\wedge\Re(z)\wedge\forall y\mathrel{\dot\in} z.\Re(gy)\\ &\to \exists g,z.x=z\wedge f=g\wedge\Re(z)\wedge\forall y\mathrel{\dot\in} z.\Re(gy)\\ &\to \Re(x)\wedge\forall y\mathrel{\dot\in} x.\Re(fy)\end{aligned}$$

For this argument, the uniqueness of generators ($\mathcal{L}_{\mathsf{EM}}$-$\mathsf{UG}$) is essential.

3 Accessible Parts in NEM

For the proof-theoretic analysis of NEM, the crucial property is the possibility of defining *accessible parts*. This will be used in the next section to embed the theory $\mathsf{ID}_1^{\mathsf{acc}}$ in NEM.

Let us introduce the following abbreviation:

$$\mathsf{Closed}(a,b,\varphi) := \forall x\mathrel{\dot\in} a.(\forall y\mathrel{\dot\in} a.(y,x)\mathrel{\dot\in} b\to\varphi(y))\to\varphi(x).$$

If b is a name for a binary relation, then $\mathsf{Closed}(a,b,\varphi)$ expresses that φ holds for all elements $c\mathrel{\dot\in} a$ if it holds for all predecessors of c in a with respect to the relation named by b.

Using this abbreviation we can state the following proposition which is the essential step of the embedding of $\mathsf{ID}_1^{\mathsf{acc}}$.

Theorem 1. *There exists a formula* $\mathsf{Acc}(a,b,x)$ *such that* NEM *proves for arbitrary formulae* $\varphi(x)$*:*

($\mathsf{Acc}.1$) $\Re(a)\wedge\Re(b)\to\mathsf{Closed}(a,b,\mathsf{Acc}(a,b,\cdot))$,
($\mathsf{Acc}.2$) $\Re(a)\wedge\Re(b)\wedge\mathsf{Closed}(a,b,\varphi)\to\forall x.\mathsf{Acc}(a,b,x)\to\varphi(x)$.

Proof. Let us assume $\Re(a,A)$ and $\Re(b,B)$. We set $A_x=\{y\in A|(y,x)\in B\}$, i.e. the subset of A consisting of all B-predecessors of x. By elementary comprehension, there exists a closed term pd so that $\Re(\mathsf{pd}\,(a,b,x),A_x)$.

By use of the recursion theorem, we can define a term f satisfying the equation:

$$\mathsf{f}\,(a,b,c)\simeq\mathsf{j}\,(\mathsf{pd}\,(a,b,c),\lambda y.\mathsf{f}(a,b,y)).\qquad(\star)$$

Hence, f maps an element $c\in A$ to the disjoint union of all f-images of B-predecessors of c. Using f, we define the formula Acc in the following way:

$$\mathsf{Acc}(a,b,c) := c\mathrel{\dot\in} a\wedge\Re(\mathsf{f}\,(a,b,c)).$$

If $\mathsf{Acc}(a,b,c)$ holds we say that "c is accessible". The idea of its definition is the following. $\mathsf{pd}\,(a,b,c)$ is the name of the set A_c which contains of all B-predecessors of c in A. Using join, we associate this set with a set of elements which can be proven to be names if $\mathsf{f}\,(a,b,c)$ is a name. This trick allows us to encode *arbitrary* objects of our language by *names*, and then name induction can be used to prove the required properties.

(Acc.1) To show $\mathsf{Closed}(a, b, \mathsf{Acc}(a, b, \cdot))$, we choose an element c of A such that

$$\forall y \mathrel{\dot\in} a.(y, c) \mathrel{\dot\in} b \to \mathsf{Acc}(a, b, y).$$

The definition of pd yields

$$\forall y.y \mathrel{\dot\in} \mathsf{pd}\,(a, b, c) \to \mathsf{Acc}(a, b, y).$$

This implies by the definition of Acc that

$$\forall y.y \mathrel{\dot\in} \mathsf{pd}\,(a, b, c) \to \Re(\mathsf{f}\,(a, b, y)).$$

From the axioms about join, we obtain

$$\Re(\mathsf{j}\,(\mathsf{pd}\,(a, b, c), \mathsf{f})).$$

By the equation $(\star)$, this means $\Re(\mathsf{f}\,(a, b, c))$. Together with the assumption $c \mathrel{\dot\in} a$ we have $\mathsf{Acc}(a, b, c)$. Since c was chosen arbitrarily, the proof of $\mathsf{Closed}(a, b, \mathsf{Acc})$ is completed.

(Acc.2) To prove the second assertion we first show two auxiliary statements (A) and (B).
(A) says that if c is accessible, then all its b predecessors are accessible, too.

$$\mathsf{Acc}(a, b, c) \to (\forall x \mathrel{\dot\in} \mathsf{pd}\,(a, b, c).\mathsf{Acc}(a, b, x)). \tag{A}$$

Assuming $\mathsf{Acc}(a, b, c)$, we get by $(\star)$ that $\Re(\mathsf{j}\,(\mathsf{pd}\,(a, b, c), \lambda y.\mathsf{f}(a, b, y)))$ holds. Then $\forall x \mathrel{\dot\in} \mathsf{pd}\,(a, b, c).\Re(\mathsf{f}(a, b, x))$ is a consequence of Lemma 2 about name strictness. To complete the proof of (A), we have to check that $\forall x \mathrel{\dot\in} \mathsf{pd}\,(a, b, c).x \mathrel{\dot\in} a$, which immediately follows from the definition of pd.

In order to formulate the assertion (B), we define an additional formula $\psi_\varphi(u, v, w)$ depending on a formula $\varphi(x)$ which will be used as induction formula in the schema of name induction. Using the definition of f, here we "replace" an arbitrary objects by their associated names.

$$\psi_\varphi(a, b, u) := \forall y.\mathsf{Acc}(a, b, y) \wedge \mathsf{f}\,(a, b, y) = u \to \varphi(y).$$

Now, the statement (B) reads as

$$\mathsf{Closed}(a, b, \varphi) \wedge \mathcal{C}(\psi_\varphi(a, b, \cdot), u) \to \psi_\varphi(a, b, u). \tag{B}$$

For the proof of (B), we assume $\mathsf{Closed}(a, b, \varphi) \wedge \mathcal{C}(\psi_\varphi(a, b, \cdot), u)$ and $\mathsf{Acc}(a, b, c) \wedge \mathsf{f}\,(a, b, c) = u$, from which we have to show $\varphi(c)$. From the last assumption, we get by $(\star)$:

$$u = \mathsf{j}\,(\mathsf{pd}\,(a, b, c), \lambda y.\mathsf{f}(a, b, y)).$$

Uniqueness of generators and clause (5) of $\mathcal{C}(\psi_\varphi(a, b, \cdot), u)$ yield

$$\forall x \mathrel{\dot\in} \mathsf{pd}\,(a, b, c).\psi_\varphi(a, b, \mathsf{f}(a, b, x))).$$

By the definition of ψ_φ, this reads

$$\forall x \,\dot{\in}\, \mathsf{pd}\,(a,b,c).\forall y.\mathsf{Acc}(a,b,y) \wedge \mathsf{f}\,(a,b,y) = \mathsf{f}(a,b,x) \rightarrow \varphi(y).$$

Choosing x for y, we get

$$\forall x \,\dot{\in}\, \mathsf{pd}\,(a,b,c).\mathsf{Acc}(a,b,x) \rightarrow \varphi(x).$$

Assuming $\mathsf{Acc}(a,b,c)$, we obtain by (A) that $\forall x \,\dot{\in}\, \mathsf{pd}\,(a,b,c).\mathsf{Acc}(a,b,x)$ holds. So we have

$$\forall x \,\dot{\in}\, \mathsf{pd}\,(a,b,c).\varphi(x).$$

But this is the premise of the assumption $\mathsf{Closed}(a,b,\varphi)$ and we get $A(c)$. Thus, (B) is proven.

To prove the second assertion (Acc.2), we now take an arbitrary formula $\varphi(x)$ and assume $\mathsf{Closed}(a,b,\varphi)$ and $\mathsf{Acc}(a,b,x)$. For the first assumption (B) yields

$$\forall y.\mathcal{C}(\psi_\varphi(a,b,\cdot),y) \rightarrow \psi_\varphi(a,b,y).$$

This is just the premise of name induction for $\psi_\varphi(a,b,y)$ and we get from $(\mathcal{L}_{\mathsf{EM}}\text{-}\mathsf{I}_\Re)$

$$\forall y.\Re(y) \rightarrow \psi_\varphi(a,b,y).$$

By the definition of $\psi_\varphi(a,b,y)$, this is

$$\forall y.\Re(y) \rightarrow \forall x.\mathsf{Acc}(a,b,x) \wedge \mathsf{f}\,(a,b,x) = y \rightarrow \varphi(x).$$

Since the assumption $\mathsf{Acc}(a,b,x)$ implies $\Re(\mathsf{f}\,(a,b,x))$, we can choose y as $\mathsf{f}\,(a,b,x)$ and all premises are satisfied. Therefore we finally obtain the required result $\varphi(x)$.

In this proof we followed the presentation of the corresponding proof in [JKS0x], where the principle of *inductive generation* is verified in the presence of *universes*.

4 Modelling $\mathsf{ID}_1^{\mathsf{acc}}$ in NEM

To show the lower bound of NEM, we will embed the theory $\mathsf{ID}_1^{\mathsf{acc}}$ of *accessibility elementary inductive definitions*, cf. [BFPS81,Can96]. Let $\mathcal{L}_1$ be the language of Peano arithmetic. In order to obtain $\mathcal{L}_{\mathsf{ID}}$, we extend this language by adding new unary predicate symbols $\mathcal{P}_\varphi$ for every formula $\varphi(x,y)$ of $\mathcal{L}_1$ containing two distinct free variables. For the definition of $\mathsf{ID}_1^{\mathsf{acc}}$, we extend the axioms of PA to the new language, including formulae induction for arbitrary $\mathcal{L}_{\mathsf{ID}}$ formulae, and add for each new predicate symbol $\mathcal{P}_\varphi$ and each $\mathcal{L}_{\mathsf{ID}}$ formula ψ the following two axioms:

($\mathsf{ID}_1^{\mathsf{acc}}$.1) $\quad \forall x.(\forall y.\varphi(x,y) \rightarrow \mathcal{P}_\varphi(y)) \rightarrow \mathcal{P}_\varphi(x)$

($\mathsf{ID}_1^{\mathsf{acc}}$.2) $\quad (\forall x.(\forall y.\varphi(x,y) \rightarrow \psi(y)) \rightarrow \psi(x)) \rightarrow \forall x.\mathcal{P}_\varphi(x) \rightarrow \psi(x)$

It is well-known that Peano arithmetic can be embedded in $\mathsf{EETJ}+(\mathcal{L}_{\mathsf{EM}}\text{-}\mathsf{I}_{\mathsf{N}})$, indeed in its applicative fragment $\mathsf{BON}+(\mathcal{L}_{\mathsf{EM}}\text{-}\mathsf{I}_{\mathsf{N}})$, using an interpretation $\cdot^N$, cf. [FJ93]. This interpretation translates formulae of $\mathcal{L}_1$ into elementary formulae of $\mathcal{L}_{\mathsf{EM}}$. Thus, by elementary comprehension we get for every binary formulae $\varphi(x,y)$ of $\mathcal{L}_1$ a name t_{φ^N} for the corresponding type, i.e. EETJ proves that t_{φ^N} is a name for $\{(x,y)|x\in\mathsf{N}\wedge y\in\mathsf{N}\wedge\varphi^N(x,y)\}$. These names will be employed in the proof of the following theorem to represent the binary relations which are used in the definition of $\mathsf{ID}_1^{\mathsf{acc}}$.

Theorem 2. *There exists a translation $\cdot^N$ from $\mathcal{L}_{\mathsf{ID}}$ to $\mathcal{L}_{\mathsf{EM}}$ such that*

$$\mathsf{ID}_1^{\mathsf{acc}}\vdash\varphi\quad\Rightarrow\quad\mathsf{NEM}\vdash\varphi^N$$

for all $\mathcal{L}_{\mathsf{ID}}$ formulae φ.

Proof. To interpret $\mathsf{ID}_1^{\mathsf{acc}}$ in NEM we extend the translation $\cdot^N$ by setting

$$[\mathcal{P}_\varphi(x)]^N:=\mathsf{Acc}(\mathsf{nat},t_{\varphi^N},x),$$

where $\mathsf{Acc}(x,y,z)$ is defined as in Theorem 1. Then the proof runs by induction on the length of the derivation of $\mathsf{ID}_1^{\mathsf{acc}}\vdash\varphi$. In addition to the embedding of PA in EETJ, we need only to check the axioms for the new predicate symbols. The translation of $(\mathsf{ID}_1^{\mathsf{acc}}.1)$ reads as

$$\begin{aligned}&[\forall x.(\forall y.\varphi(x,y)\to\mathcal{P}_\varphi(y))\to\mathcal{P}_\varphi(x)]^N\\ \leftrightarrow\ &\forall x\,\dot\in\,\mathsf{nat}.(\forall y\,\dot\in\,\mathsf{nat}.\varphi^N(x,y)\to\mathsf{Acc}(\mathsf{nat},t_{\varphi^N},y))\to\mathsf{Acc}(\mathsf{nat},t_{\varphi^N},x)\\ \leftrightarrow\ &\mathsf{Closed}(\mathsf{nat},t_{\varphi^N},\mathsf{Acc}(\mathsf{nat},t_{\varphi^N},\cdot)).\end{aligned}$$

Since the last line is an instance of (Acc.1) of Theorem 1, this axiom is verified. In the same way, $(\mathsf{ID}_1^{\mathsf{acc}}.2)^N$ follows from (Acc.2):

$$\begin{aligned}&[(\forall x.(\forall y.\varphi(x,y)\to\psi(y))\to\psi(x))\to\forall x.\mathcal{P}_\varphi(x)\to\psi(x)]^N\\ \leftrightarrow\ &(\forall x\,\dot\in\,\mathsf{nat}.(\forall y\,\dot\in\,\mathsf{nat}.\varphi^N(x,y)\to\psi^N(y))\to\psi^N(x))\\ &\qquad\to\forall x\,\dot\in\,\mathsf{nat}.\mathsf{Acc}(\mathsf{nat},t_{\varphi^N},x)\to\psi^N(x)\\ \leftrightarrow\ &\mathsf{Closed}(\mathsf{nat},t_{\varphi^N},\psi^N)\to\forall x.\mathsf{Acc}(\mathsf{nat},t_{\varphi^N},x)\to\psi^N(x).\end{aligned}$$

The last line is an instance of (Acc.2), and we have finished the embedding of $\mathsf{ID}_1^{\mathsf{acc}}$.

5 Modelling NEM in ID_1

In this section, we embed NEM in the theory ID_1 of non-iterated inductive definitions. This extension of Peano arithmetic postulates the existence of least fixed points for positive arithmetical operator forms. These are formulae $\varphi(R,x)$ in the language $\mathcal{L}_1$ with one additional relation symbol R that has only positive occurrences in φ. The language of ID_1 is $\mathcal{L}_1$ extended by new predicate symbols

$\mathcal{P}_\varphi$ for each positive operator form $\varphi(R, x)$. As axioms, we choose those of PA, including formulae induction extended to the new language and the following two principles for each new predicate symbol $\mathcal{P}_\varphi$ and arbitrary formulae ψ:

$$(\mathsf{ID}_1.1) \qquad \forall x.\varphi(\mathcal{P}_\varphi, x) \to \mathcal{P}_\varphi(x)$$

$$(\mathsf{ID}_1.2) \qquad (\forall x.\varphi(\psi/R, x) \to \psi(x)) \to \forall x.\mathcal{P}_\varphi(x) \to \psi(x)$$

Here $\varphi(\psi/R, x)$ denotes the result of substituting any occurrence of $R(t)$ in φ by $\psi(t/x)$.

In [Fef75], Feferman presented an inductive model construction for explicit mathematics. Beeson showed in [Bee85] that for the system $\mathsf{EETJ} + (\mathcal{L}_{\mathsf{EM}}\text{-}\mathsf{I}_{\mathsf{N}})$ this construction can be carried out in the theory $\widehat{\mathsf{ID}}_1$, cf. also [Mar94, MS98]. This theory stating only the existence of (not necessarily least) fixed points of positive arithmetical operator forms can be obtained from ID_1 by replacing the axioms $(\mathsf{ID}_1.1)$ and $(\mathsf{ID}_1.2)$ by

$$(\widehat{\mathsf{ID}}_1) \qquad \forall x.\varphi(\mathcal{P}_\varphi, x) \leftrightarrow \mathcal{P}_\varphi(x).$$

In fact, we can use Beeson's formalization for the analysis of NEM using, in addition, the induction principle of ID_1 to verify name induction $(\mathcal{L}_{\mathsf{EM}}\text{-}\mathsf{I}_{\Re})$. The only differences are the adaption to the finite axiomatization of elementary comprehension and the (trivial) verification of uniqueness of generators $(\mathcal{L}_{\mathsf{EM}}\text{-}\mathsf{UG})$ which was not part of the original formulation of EETJ.

We start with a standard interpretation $\cdot^\star$ of the applicative structure using the relation $App(x, y, z) := \{x\}(y) \simeq y$ in the sense of ordinary recursion, cf. [FJ93]. Here, the constants of $\mathcal{L}_{\mathsf{EM}}$ are interpreted by numerals of $\mathcal{L}_1$ coding appropriate number-theoretic functions satisfying the axioms of EETJ. With respect to the generators we have to choose numerals according to the following codes which will be used for the interpretation of the type structure:

- $\langle 1\rangle$ codes the type of numerals,
- $\langle 2\rangle$ codes the type of pairs with identical elements,
- $\langle 3, a\rangle$ codes the complement of the type coded by a,
- $\langle 4, a, b\rangle$ codes the intersection of the two types coded by a and b,
- $\langle 5, a\rangle$ codes the domain of a function given as a type of ordered pairs coded by a
- $\langle 6, f, a\rangle$ codes the inverse images of f, i.e. the type of all individuals x with fx is an element of the type coded by a,
- $\langle 7, a, f\rangle$ codes the join of f over the type coded by a.

By choosing the codes for the generators according to these conditions, the axioms about uniqueness of generators are obviously satisfied.

To interpret the second order part of NEM we define three relations Typ, In and $\overline{\mathsf{In}}$, using appropriate operator forms. The meaning of these predicates and their relation to $\mathcal{L}_{\mathsf{EM}}$ is as follows. Let s, t be terms of ID_1 interpreting types S, T of $\mathcal{L}_{\mathsf{EM}}$, respectively, and let r be the interpretation of an arbitrary $\mathcal{L}_{\mathsf{EM}}$ term, then we have:

- $\mathsf{Typ}(t)$ represents that t is a code of a type.
- $\mathsf{In}(r,t)$ interprets the formula $r \in T$.
- $\overline{\mathsf{In}}(r,t)$ holds for $\neg r \in T$.
- We have to introduce the relation $\overline{\mathsf{In}}$ in order to guarantee that the defining operator forms are positive. As a consequence, we have to prove that $\mathsf{In}(r,t)$ is equivalent to $\neg\overline{\mathsf{In}}(r,t)$.
- $T = S$ is interpreted by $\mathsf{Typ}(t) \wedge \mathsf{Typ}(s) \wedge \forall x.\mathsf{In}(x,t) \leftrightarrow \mathsf{In}(x,s)$, i.e. as extensional equality.
- $\Re(t,S)$ is also modelled by $\mathsf{Typ}(t) \wedge \mathsf{Typ}(s) \wedge \forall x.\mathsf{In}(x,t) \leftrightarrow \mathsf{In}(x,s)$.

In order to define $\mathsf{Typ}(x)$, $\mathsf{In}(x,y)$ and $\overline{\mathsf{In}}(x,y)$ we need some coding. Let us use $\varphi^0(x)$, $\varphi^1(x,y)$ and $\varphi^2(x,y)$ as abbreviations for $\varphi(\langle 0,x\rangle)$, $\varphi(\langle 1,\langle x,y\rangle\rangle)$ and $\varphi(\langle 2,\langle x,y\rangle\rangle)$, respectively. With this notation we can define $\mathsf{Typ}(x)$, $\mathsf{In}(x,y)$ and $\overline{\mathsf{In}}(x,y)$ as the "projections" $\mathcal{P}^0_{\boldsymbol{\varphi}}(x)$, $\mathcal{P}^1_{\boldsymbol{\varphi}}(x,y)$ and $\mathcal{P}^2_{\boldsymbol{\varphi}}(x,y)$ of the fixed point $\mathcal{P}_{\boldsymbol{\varphi}}$ of the positive operator form:

$$\begin{aligned}\boldsymbol{\varphi}(\psi,z) := {} & (\exists y.z = \langle 0,y\rangle \wedge \mathcal{C}_{\mathsf{Typ}}(\psi,y)) \vee \\ & (\exists x,y.z = \langle 1,\langle x,y\rangle\rangle \wedge \mathcal{C}_{\mathsf{In}}(\psi,x,y)) \vee \\ & (\exists x,y.z = \langle 2,\langle x,y\rangle\rangle \wedge \mathcal{C}_{\overline{\mathsf{In}}}(\psi,x,y))\end{aligned}$$

with the following closure conditions (where it is helpful to keep in mind the intended meanings of ψ^0, ψ^1 and ψ^2, namely Typ, In and $\overline{\mathsf{In}}$, respectively). $\mathcal{C}_{\mathsf{Typ}}(\psi,z)$ is the disjunction of the following clauses:

- $z = \langle 1\rangle$,
- $z = \langle 2\rangle$,
- $\exists x.z = \langle 3,x\rangle \wedge \psi^0(x)$,
- $\exists x,y.z = \langle 4,x,y\rangle \wedge \psi^0(x) \wedge \psi^0(x)$,
- $\exists x.z = \langle 5,x\rangle \wedge \psi^0(x)$,
- $\exists f,x.z = \langle 6,f,x\rangle \wedge \psi^0(x)$,
- $\exists f,x.z = \langle 7,x,f\rangle \wedge \psi^0(x) \wedge \forall y.\neg\psi^2(y,x) \to \psi^0(\{f\}(y))$.

$\mathcal{C}_{\mathsf{In}}(\psi,u,z)$ is the disjunction of the following clauses:

- $z = \langle 0\rangle$,
- $z = \langle 1\rangle \wedge \exists y.u = \langle y,y\rangle$,
- $\exists x.z = \langle 2,x\rangle \wedge \psi^0(x) \wedge \psi^2(u,x)$,
- $\exists x,y.z = \langle 4,x,y\rangle \wedge \psi^0(x) \wedge \psi^0(x) \wedge \psi^1(u,x) \wedge \psi^1(u,y)$,
- $\exists x.z = \langle 5,x\rangle \wedge \psi^0(x) \wedge \exists v.\psi^1(\langle u,v\rangle,x)$,
- $\exists f,x.z = \langle 6,f,x\rangle \wedge \psi^0(x) \wedge \psi^1(\{f\}(u),x)$,
- $\exists f,x.z = \langle 7,x,f\rangle \wedge \psi^0(x) \wedge (\forall y.\neg\psi^2(y,x) \to \psi^0(\{f\}(y))) \wedge \exists v,w.u = \langle v,w\rangle \wedge \psi^1(v,x) \wedge \psi^1(w,\{f\}(v))$.

The defining clauses for $\mathcal{C}_{\overline{\mathsf{In}}}$ are analogous, also containing positive occurrences of ψ only.

Without the leastness property for the fixed point defined by $\boldsymbol{\varphi}$ we cannot prove that In and $\overline{\mathsf{In}}$ are complementary. Hence, for embedding $\mathsf{EETJ}+(\mathcal{L}_{\mathsf{EM}}\text{-}\mathsf{I}_\mathsf{N})$ in $\widehat{\mathsf{ID}}_1$ one has to make use of Aczel's trick of sorting out all codes a for types where $\mathsf{In}(\cdot,a)$ is not the complement of $\overline{\mathsf{In}}(\cdot,a)$. However, in ID_1 the leastness condition allows for a direct proof that In and $\overline{\mathsf{In}}$ are complements, cf. [Bee85].

Lemma 3. $\mathsf{ID}_1 \vdash \mathsf{Typ}(y) \to \forall x.\mathsf{In}(x,y) \leftrightarrow \neg\overline{\mathsf{In}}(x,y)$.

Theorem 3. NEM *can be embedded in* ID_1.

Proof. The interpretation $\cdot^\star$ is chosen according to the remarks above. The verification of the axioms of EETJ and the induction schema ($\mathcal{L}_{\mathsf{EM}}$-$\mathsf{I}_{\mathsf{N}}$) is straightforward, cf. [Bee85] and [Mar94]. It only remains to check the principle of name induction,

$$(\mathcal{L}_{\mathsf{EM}}\text{-}\mathsf{I}_{\Re}) \qquad (\forall x.\mathcal{C}(\chi,x) \to \chi(x)) \to \forall x.\Re(x) \to \chi(x).$$

This can be derived from the leastness principle for $\mathcal{P}_{\boldsymbol{\varphi}}$

$$(\forall z.\boldsymbol{\varphi}(\psi,z) \to \psi(z)) \to \forall z.\mathcal{P}_{\boldsymbol{\varphi}}(z) \to \psi(z)$$

by choosing a formula $\psi(z)$ so that

$$\begin{aligned} \psi(\langle 0,x\rangle) &\leftrightarrow \chi^\star(x),\\ \psi(\langle 1,\langle x,y\rangle\rangle) &\leftrightarrow \mathsf{In}(x,y),\\ \psi(\langle 2,\langle x,y\rangle\rangle) &\leftrightarrow \overline{\mathsf{In}}(x,y),\\ \psi(z) &\leftrightarrow 0=0 \qquad \text{for every other argument } z. \end{aligned}$$

Starting from the premise $[\forall x.\mathcal{C}(\chi,x) \to \chi(x)]^\star$ we obtain $(\forall z.\boldsymbol{\varphi}(\psi,z) \to \psi(z))$: assume $\boldsymbol{\varphi}(\psi,z)$ holds with $z = \langle 0,x\rangle$ for some x. Then we get $\mathcal{C}_{\mathsf{Typ}}(\psi,x)$ which implies $[\mathcal{C}(\chi,x)]^\star$. So $\chi^\star(x)$ follows by our premise and $\psi(\langle 0,x\rangle)$ holds by the definition of ψ. If $\boldsymbol{\varphi}(\psi,z)$ holds and there is no x with $z = \langle 0,x\rangle$, then $\psi(z)$ is trivially fulfilled. Hence we conclude by the leastness condition for $\mathcal{P}_{\boldsymbol{\varphi}}$ that $\forall z.\mathcal{P}_{\boldsymbol{\varphi}}(z) \to \psi(z)$ holds. Let z be $\langle 0,x\rangle$, then we have $\mathcal{P}_{\boldsymbol{\varphi}}(\langle 0,x\rangle) \to \psi(\langle 0,x\rangle)$ which reads as $\mathsf{Typ}(x) \to \chi^\star(x)$. Because $\Re(x)$ is interpreted as $\mathsf{Typ}(x)$ we are finished.

This theorem, together with Theorem 2 and the well-known proof-theoretic equivalence of $\mathsf{ID}_1^{\mathsf{acc}}$ and ID_1, yields the final result:

Theorem 4. *The theory* NEM *of explicit mathematics with name induction is proof-theoretically equivalent to* ID_1*, and its proof-theoretic ordinal is the Bachmann-Howard ordinal.*

References

AC96. Martín Abadi and Luca Cardelli. *A Theory of Objects*. Springer, 1996.

Bee85. Michael Beeson. *Foundations of Constructive Mathematics*. Ergebnisse der Mathematik und ihrer Grenzgebiete; 3.Folge, Bd. 6. Springer, Berlin, 1985.

BFPS81. Wilfried Buchholz, Solomon Feferman, Wolfram Pohlers, and Wilfried Sieg. *Iterated Inductive Definitions and Subsystems of Analysis: Recent Proof-Theoretical studies*, volume 897 of *Lecture Notes in Mathematics*. Springer-Verlag, 1981.

Can96. Andrea Cantini. *Logical Frameworks for Truth and Abstraction*, volume 135 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 1996.

Fef70. Solomon Feferman. Formal theories for transfinite iterations of generalized inductive definitions and some subsystems of analysis. In A. Kino, J. Myhill, and R. Vesley, editors, *Intuitionismus and Proof Theory*, pages 303–326. North Holland, Amsterdam, 1970.

Fef75. Solomon Feferman. A language and axioms for explicit mathematics. In J. Crossley, editor, *Algebra and Logic*, volume 450 of *Lecture Notes in Mathematics*, pages 87–139. Springer, 1975.

Fef79. Solomon Feferman. Constructive theories of functions and classes. In M. Boffa, D. van Dalen, and K. McAloon, editors, *Logic Colloquium 78*, pages 159–224. North–Holland, Amsterdam, 1979.

Fef88. Solomon Feferman. Hilbert's program relativized: Proof-theoretical and foundational reductions. *Journal of Symbolic Logic*, 53(2):364–384, 1988.

Fef90. Solomon Feferman. Polymorphic typed lambda-calculus in a type-free axiomatic framework. In W. Sieg, editor, *Logic and Computation*, volume 106 of *Contemporary Mathematics*, pages 101–136. American Mathematical Society, 1990.

Fef91. Solomon Feferman. Logics for termination and correctness of functional programs. In Y. Moschovakis, editor, *Logic from Computer Sciences*, pages 95–127. Springer, 1991.

Fef92. Solomon Feferman. Logics for termination and correctness of functional programs II: Logics of strength PRA. In P. Aczel, H. Simmons, and S. S. Wainer, editors, *Proof Theory*, pages 195–225. Cambridge University Press, 1992.

Fef0x. Solomon Feferman. Does reductive proof theory have a viable rationale? *Erkenntnis*, 200x. To appear.

FJ93. Solomon Feferman and Gerhard Jäger. Systems of explicit mathematics with non-constructive μ-operator. Part I. *Annals of Pure and Applied Logic*, 65(3):243–263, 1993.

FJ96. Solomon Feferman and Gerhard Jäger. Systems of explicit mathematics with non-constructive μ-operator. Part II. *Annals of Pure and Applied Logic*, 79:37–52, 1996.

GR94. Ed Griffor and Michael Rathjen. The strength of some Martin-Löf type theories. *Archive for Mathematical Logic*, 33:347–385, 1994.

Jäg88. Gerhard Jäger. Induction in the elementary theory of types and names. In E. Börger, H. Kleine Büning, and M.M. Richter, editors, *Computer Science Logic '87*, volume 329 of *Lecture Notes in Computer Science*, pages 118–128. Springer, 1988.

JKS99. Gerhard Jäger, Reinhard Kahle, and Thomas Strahm. On applicative theories. In A. Cantini, E. Casari, and P. Minari, editors, *Logic and Foundation of Mathematics*, pages 83–92. Kluwer, 1999.

JKS0x. Gerhard Jäger, Reinhard Kahle, and Thomas Studer. Universes in explicit mathematics. 200x. Submitted.

Kre63. Georg Kreisel. Generalized inductive definitions. Technical report, Stanford Report, 1963.

Mar94. Markus Marzetta. *Predicative Theories of Types and Names*. Dissertation, Universität Bern, Institut für Informatik und angewandte Mathematik, 1994.

MS98. Markus Marzetta and Thomas Strahm. The μ quantification operator in explicit mathematics with universes and iterated fixed point theories with ordinals. *Archive for Mathematical Logic*, 37:391–413, 1998.

Pal98. Erik Palmgren. On universes in type theory. In G. Sambin and J. Smith, editors, *Twenty Five Years of Constructive Type Theory*, pages 191–204. Oxford University Press, 1998.

Poh89. Wolfram Pohlers. *Proof Theory*, volume 1407 of *Lecture Notes in Mathematics*. Springer, 1989.

Stä97. Robert Stärk. Call-by-value, call-by-name and the logic of values. In D. van Dalen and M. Bezem, editors, *Computer Science Logic CSL '96: Selected Papers*, volume 1258 of *Lecture Notes in Computer Science*, pages 431–445. Springer, 1997.

Stä98. Robert Stärk. Why the constant 'undefined'? Logics of partial terms for strict and non-strict functional programming languages. *Journal of Functional Programming*, 8(2):97–129, 1998.

Stu0x. Thomas Studer. A semantics for $\lambda_{str}^{\{\}}$: a calculus with overloading and late-binding. *Journal of Logic and Computation*, 200x. To appear.

TvD88. Anne Troelstra and Dirk van Dalen. *Constructivism in Mathematics*, volume II. North Holland, Amsterdam, 1988.

On the Complexity of Explicit Modal Logics

Roman Kuznets*

Moscow State University, Moscow 119899, Russia

Abstract. Explicit modal logic was introduced by S. Artemov. Whereas the traditional modal logic uses atoms $\Box F$ with a possible semantics "F is provable", the explicit modal logic deals with atoms of form $t{:}F$, where t is a *proof polynomial* denoting a specific proof of a formula F. Artemov found the explicit modal logic $\mathcal{LP}$ in this new format and built an algorithm that recovers explicit proof polynomials corresponding to modalities in every derivation in K. Gödel's modal provability calculus $\mathcal{S}4$. In this paper we study the complexity of $\mathcal{LP}$ as well as the complexity of explicit counterparts of the modal logics $\mathcal{K}$, $\mathcal{D}$, $\mathcal{T}$, $\mathcal{K}4$, $\mathcal{D}4$ found by V. Brezhnev. The main result: the satisfiability problem for each of these explicit modal logics belongs to the class Σ_2^p of the polynomial hierarchy. Similar problem for the original modal logics is known to be PSPACE-complete. Therefore, explicit modal logics have much better upper complexity bounds than the original modal logics.

1 Introduction and Main Definitions

The idea to describe provability by means of modal logic was formulated by K. Gödel in [6]. He axiomatized the general properties of provability in the modal language and obtained the modal logic coinciding with $\mathcal{S}4$. However the problem of finding the exact provability semantics for $\mathcal{S}4$ remained open.

The explicit logic of proofs $\mathcal{LP}$ formulated in terms of the predicate "t is a proof of A" was introduced by S. Artemov in [2]. It incorporates proofs into propositional language by means of *proof polynomials* constructed with the help of elementary computable operations corresponding to *modus ponens*, proof-checking and non-deterministical choice. $\mathcal{LP}$ is supplied with the appropriate arithmetical semantics and is proved to be complete with respect to this semantics (cf. [2]). $\mathcal{LP}$ is proved to be an explicit counterpart of logic of informal provability $\mathcal{S}4$. Namely, $\mathcal{LP}$ is sufficient to realize the whole $\mathcal{S}4$ by assigning explicit proof polynomials to the occurrences of $\Box$ in $\mathcal{S}4$-derivation (cf. [2]). So Logic of Proofs $\mathcal{LP}$ provides $\mathcal{S}4$ with the intended provability reading.

Explicit analogues of modal logics weaker than $\mathcal{S}4$ (in particular of $\mathcal{K}$, $\mathcal{D}$, $\mathcal{T}$, $\mathcal{K}4$, $\mathcal{D}4$) were introduced by V. Brezhnev in [5]. He suggested axiomatization for them and proved that they suffice to realize the corresponding modal logics (in the same way as $\mathcal{LP}$ realizes $\mathcal{S}4$).

* The author is partially supported by the grant DAAH04–96–1–0341, by DAPRA under program LPE, project 34145.

P. Clote and H. Schwichtenberg (Eds.): CSL 2000, LNCS 1862, pp. 371–383, 2000.

Decidability of $\mathcal{LP}$ was proved by A. Mkrtychev in [7]. In this paper we adapt this algorithm to the explicit logics introduced in [5] and evaluate its complexity. It turned out that the derivability problem for all of these logics belongs to the class Π_2^p in the polynomial hierarchy while all the corresponding modal logics are known to be PSPACE-complete.

In this section we will formulate the language of explicit modal logics (or the language of $\mathcal{LP}$) and give axiomatics for them. In Section 2 we will describe the semantics for the explicit logics and prove their completeness with respect to it. This semantics for $\mathcal{LP}$ was suggested by Mkrtychev in [7]. We adapt it for all the other explicit logics. Finally, in Section 3 we describe the decision algorithm and evaluate its complexity.

Let us describe the modal logics we deal with. First, we give the full list of modal axioms:

A0 Axioms of classical propositional logic in the monomodal language

AN $\Box(F \to G) \to (\Box F \to \Box G)$ *normality*

AD $\Box\bot \to \bot$ *seriality*

AT $\Box F \to F$ *reflexivity*

A4 $\Box F \to \Box\Box F$ *transitivity*

The minimal logic $\mathcal{K}$ contains axioms **A0** and **AN** and the rules of inference *modus ponens* and necessitation $\dfrac{F}{\Box F}$. All the other logics are extensions of $\mathcal{K}$ with the additional axioms: $\mathcal{D} = \mathcal{K} + \mathbf{AD}$, $\mathcal{T} = \mathcal{K} + \mathbf{AT}$, $\mathcal{K}4 = \mathcal{K} + \mathbf{A4}$, $\mathcal{D}4 = \mathcal{D} + \mathbf{A4}$ and $\mathcal{S}4 = \mathcal{T} + \mathbf{A4}$.

Now we turn to the description of the explicit modal logics (cf. [2], [5]). They are formulated in the language of $\mathcal{LP}$ that contains proof variables x_i, proof constants a_i and operations on proofs (binary $\cdot$, $+$ and monadic $!$); sentence letters S_i, boolean connectives, boolean constant $\bot$, and the binary proof operator (*polynomial*):(*formula*). Proofs are represented by polynomials generated from proof variables and constants by means of operations on proofs. Formulae are constructed from the sentence letters and boolean constants in the usual way with the additional rule: if F is a formula and t is a polynomial then $t : F$ is a formula. Let *SVar* stand for the set of all sentence letters, *Pn* for the set of polynomials and *Fm* for the set of formulae.

Now we are going to define the following explicit logics:

$$\mathcal{LP}(\mathcal{K}), \mathcal{LP}(\mathcal{D}), \mathcal{LP}(\mathcal{T}), \mathcal{LP}(\mathcal{K}4), \mathcal{LP}(\mathcal{D}4), \mathcal{LP}(\mathcal{S}4). \tag{1}$$

In what follows by explicit logic we mean any of these six logics. As before, first we give the list of explicit axioms (cf. [2], [5])

A0 Axioms of classical propositional logic in the language of $\mathcal{LP}$

A1 $t : (F \to G) \to (s : F \to (t \cdot s) : G)$

A2 $t_i : F \to (t_1 + t_2) : F$, $i = 1, 2$

A3 $t : \bot \to \bot$

A4 $t : F \to !\, t : (t : F)$

A5 $t : F \to F$

There are two explicit versions of the modal necessitation rule for an explicit logic L

$ONec$ $\dfrac{}{!^n\, a : (!^{n-1}\, a : \ldots (!\, a : (a : \mathbf{A}))\ldots)}$;

$ONec'$ $\dfrac{}{a : \mathbf{A}}$,

where a is a proof constant, $\mathbf{A}$ is an axiom of L.

All the explicit logics contain axioms **A0**, **A1**, **A2** and the rule *modus ponens*. To obtain the axiom system for them one has to add $ONec$ for $\mathcal{LP}(\mathcal{K})$; **A3** and $ONec$ for $\mathcal{LP}(\mathcal{D})$; **A5** and $ONec$ for $\mathcal{LP}(\mathcal{T})$; **A4** and $ONec'$ for $\mathcal{LP}(\mathcal{K}4)$; **A4**, **A3** and $ONec'$ for $\mathcal{LP}(\mathcal{D}4)$; **A4**, **A5** and $ONec'$ for $\mathcal{LP}(\mathcal{S}4)$.

It may be easily observed that all explicit axioms except for **A2** can be obtained by replacing $\Box$ in a corresponding modal axiom for a certain proof polynomial. So we use names of modal axioms referring to their explicit analogues. For example, we call $\mathcal{LP}(\mathcal{K}4)$, $\mathcal{LP}(\mathcal{D}4)$ and $\mathcal{LP}(\mathcal{S}4)$ transitive explicit logics since they contain the explicit axiom **A4** corresponding to the modal axiom of transitivity **A4**.

By an explicit realization r of a modal formula F we mean an assignment of proof polynomials to all occurrences of modality in F, the image of F under such a realization is denoted by F^r. Now we formulate the main result about the connection between modal logics and their explicit analogues.

Theorem 1 (Artemov). *$\mathcal{S}4 \vdash F$ iff $\mathcal{LP} \vdash F^r$ for some realization r (cf. [2]).*

Theorem 2 (Brezhnev). *Let L be one of modal logics $\mathcal{K}$, $\mathcal{D}$, $\mathcal{T}$, $\mathcal{K}4$, $\mathcal{D}4$. Then $L \vdash F$ iff $\mathcal{LP}(L) \vdash F^r$ for some realization r (cf. [5]).*

2 Semantics for Explicit Logics and Completeness Theorem

In this section we describe semantics for the logics (1). This semantics for the logic $\mathcal{LP}(\mathcal{S}4) = \mathcal{LP}$ was introduced by Mkrtychev in [7]. He also proved completeness of $\mathcal{LP}$ with respect to this semantics. In this section his results are generalized to all explicit logics.

Definition 1. *A function $* : Pn \to 2^{Fm}$ that assigns to every proof polynomial a set of $\mathcal{LP}$-formulae is called a* proof-theorem assignment *if it satisfies the following two conditions:*

1. *if $(F \to G) \in *(t)$ and $F \in *(s)$ then $G \in *(t \cdot s)$;*
2. *$*(t) \cup *(s) \subseteq *(t + s)$.*

A proof-theorem assignment is called transitive *if it satisfies in addition a transitivity condition: if $F \in *(t)$ then $(t : F) \in *(!\, t)$. A proof-theorem assignment is* serial *if $\bot \notin *(t)$ for every proof polynomial t.*

Remark 1. We do not require the sets $*(t)$ to be finite.

Definition 2. *A* model $\mathcal{M}$ *is a triple* $(v, *, \models)$*, where* v *is a truth-assignment, i. e. a mapping* $v : SVar \to \{True, False\}$*,* $*$ *is a proof-theorem assignment and* $\models$ *is a truth relation. The latter is defined in the following way.*

1. *For sentence letters* $\models S \quad \Leftrightarrow \quad v(S) = True$ *and* $\not\models \bot$*;*
2. $\models F \to G \quad \Leftrightarrow \quad \not\models F$ *or* $\models G$*;*
3. $\models t : F \quad \Leftrightarrow \quad F \in *(t)$.

Remark 2. In what follows we omit the cases concerning boolean connectives other than implication. These missing cases can be easily restored using expressions for the connectives in terms of implication and $\bot$.

Definition 3. *A model* $\mathcal{M} = (v, *, \models)$ *is called* reflexive *if* $F \in *(t)$ *implies* $\mathcal{M} \models F$ *for any polynomial* t *and any formula* F*.*

Evidently, there is a precise correspondence between conditions imposed on a proof-theorem assignment and the explicit axioms **A1**, **A2**. Transitive assignment satisfies axiom **A4** which corresponds to the transitivity modal axiom. Similarly, axiom **A3** corresponding to seriality modal axiom is true for any serial assignment. Finally, in a reflexive model axiom **A5** expressing weak reflexivity also holds.

Let us call any set of formulae introduced by the necessitation rule a Constant Specification (CS) for the logic L. Namely, for the logics $\mathcal{LP}(\mathcal{K})$, $\mathcal{LP}(\mathcal{D})$, $\mathcal{LP}(\mathcal{T})$ CS is any set of formulae of the form $!^n a{:}(!^{n-1} a{:}\ldots{:}(!\, a{:}(a{:}\mathbf{A}))\ldots)$, where a is a proof constant, $\mathbf{A}$ is an axiom of the corresponding logic. For $\mathcal{LP}(\mathcal{K}4)$, $\mathcal{LP}(\mathcal{D}4)$, $\mathcal{LP}(\mathcal{S}4)$ formulae should be of the form $a : \mathbf{A}$, where a is a proof constant, $\mathbf{A}$ is an axiom of the corresponding logic.

For any explicit logic L let CS_L denote a maximal constant specification, namely

$$CS_L = \{!^n a : \ldots : (!\, a : (a : \mathbf{A})) \mid a \text{ is a proof constant, } \mathbf{A} \text{ is an axiom of } L\}$$

if $L \in \{\mathcal{LP}(\mathcal{K}), \mathcal{LP}(\mathcal{D}), \mathcal{LP}(\mathcal{T})\}$, or

$$CS_L = \{a : \mathbf{A} \mid a \text{ is an axiom constant, } \mathbf{A} \text{ is an axiom of } L\}$$

if $L \in \{\mathcal{LP}(\mathcal{K}4), \mathcal{LP}(\mathcal{D}4), \mathcal{LP}(\mathcal{S}4)\}$.

Remark 3. The specification CS_L depends on the axiomatization chosen for propositional logic in **A0**.

Definition 4. *Let* CS *be a constant specification. A model* $\mathcal{M}$ *is called a* CS-model *if* $\mathcal{M} \models CS$*.*

Table 1. Additional conditions on L-models

Explicit logic	Proof-theorem assignment	Model
$\mathcal{LP}(\mathcal{K})$		
$\mathcal{LP}(\mathcal{D})$	serial	
$\mathcal{LP}(\mathcal{T})$		reflexive
$\mathcal{LP}(\mathcal{K}4)$	transitive	
$\mathcal{LP}(\mathcal{D}4)$	serial transitive	
$\mathcal{LP}(\mathcal{S}4)$	transitive	reflexive

Definition 5. *Let L be one of the logics (1). L-model is any CS_L-model satisfying additional conditions given in Table 1.*

The following theorem states the completeness of explicit modal logics with respect to the semantics described above.

Theorem 3 (completeness). *Let L be an explicit logic. Then*

$$L \vdash F \quad \Leftrightarrow \quad F \text{ is true in all } L\text{-models.}$$

The proof of Theorem 3 is standard, so we just give the main ideas in brief.

Definition 6. *Let L be one of the logics (1).*

The set $\mathcal{F} \subset Fm$ is called L-consistent *if $L \nvdash \neg(A_1 \wedge \ldots \wedge A_n)$ for any finite subset $\{A_1, \ldots, A_n\} \subseteq \mathcal{F}$. $\mathcal{F}$ is called* maximal L-consistent *if in addition either $F \in \mathcal{F}$ or $\neg F \in \mathcal{F}$ holds for any $\mathcal{LP}$-formula F.*

The following lemma is standard.

Lemma 1.

1. *Let $\mathcal{F}$ be an L-consistent set. Then there exists a maximal L-consistent set $\mathcal{F}'$ such that $\mathcal{F} \subseteq \mathcal{F}'$.*
2. *Any maximal L-consistent set contains L and is closed under the inference rules of L.*

Lemma 2. *Suppose $\mathcal{F}$ is a maximal L-consistent set. Then there exists an L-model $\mathcal{M}$ such that $\mathcal{M} \models \mathcal{F}$.*

Proof. In order to construct the desired model let us define the proof-theorem assignment $*(t) = \{F \in Fm \mid t : F \in \mathcal{F}\}$ for any polynomial t. It can be easily observed that $*$ is a proof-theorem assignment. Moreover, $*$ is serial for $L \in \{\mathcal{LP}(\mathcal{D}), \mathcal{LP}(\mathcal{D}4)\}$ and transitive for $L \in \{\mathcal{LP}(\mathcal{K}4), \mathcal{LP}(\mathcal{D}4), \mathcal{LP}(\mathcal{S}4)\}$.

For every sentence letter S let us put $v(S) = True$ iff $S \in \mathcal{F}$.

Let us consider the model $\mathcal{M} = (v, *, \models)$. By induction on complexity of the formula F it can be easily shown that $\mathcal{M} \models F$ iff $F \in \mathcal{F}$. At the same time, $(t : F \to F) \in L \subset \mathcal{F}$ for the reflexive logics $\mathcal{LP}(\mathcal{T})$, $\mathcal{LP}(\mathcal{S}4)$, that provides reflexivity of the model $\mathcal{M}$ for them. So $\mathcal{M}$ is an L-model. □

Proof (of Theorem 3). If $L \vdash F$ then obviously $\mathcal{M} \models F$ for any L-model $\mathcal{M}$. Suppose $L \nvdash F$. In such a case the set $\{\neg F\}$ is L-consistent and we can extend it to some maximal L-consistent set $\mathcal{F}$. By Lemma 2 there exists an L-model $\mathcal{M}$ such that $\mathcal{M} \models \mathcal{F}$, in particular $\mathcal{M} \models \neg F$. □

While dealing with reflexive logics $\mathcal{LP}(\mathcal{T})$ and $\mathcal{LP}(\mathcal{S}4)$ one have to prove reflexivity of a given model. The following notion allows avoiding this difficulty.

Definition 7. *A* pre-model $\mathcal{P}$ *is a triple* $(v, *, \models_p)$*, where* v *is a truth-assignment,* $*$ *is a proof-theorem assignment and the definition of a truth relation* $\models_p$ *is similar to* $\models$ *(see Definition 2) except for the case*

$$\models_p t : F \quad \Leftrightarrow \quad F \in *(t) \text{ and } \models_p F.$$

Definition 8. *A model* $\mathcal{M} = (v, *, \models)$ *and a pre-model* $\mathcal{P} = (v', *', \models_p)$ *are called* equivalent *if the truth relations* $\models$ *and* $\models_p$ *coincide.*

The following lemma describes correlation between the notions of a model and a pre-model.

Lemma 3. *For any reflexive model* $\mathcal{M} = (v, *, \models)$ *there exists a pre-model* $\mathcal{P} = (v', *', \models_p)$ *equivalent to it. Conversely, for any pre-model* $\mathcal{P} = (v', *', \models_p)$ *there exists a reflexive model* $\mathcal{M} = (v, *, \models)$ *equivalent to it. Moreover, if initial model (pre-model) is transitive then the resulting pre-model (model) is also transitive.*

Proof. Suppose $\mathcal{M} = (v, *, \models)$ is a reflexive model. Then the pre-model $\mathcal{P} = (v, *, \models_p)$ is equivalent to $\mathcal{M}$, i. e. $\mathcal{P} \models_p F \Leftrightarrow \mathcal{M} \models F$. Reason by induction on the complexity of F. The case of sentence letters and boolean connectives is trivial. Let $F = t : G$. If $\mathcal{P} \models_p t : G$ then $G \in *(t)$ and $\mathcal{M} \models t : G$. Conversely, if $\mathcal{M} \models t : G$ then $G \in *(t)$. The model $\mathcal{M}$ is reflexive, so $\mathcal{M} \models G$. By the induction hypothesis $\mathcal{P} \models_p G$. Thus, we obtain $\mathcal{P} \models_p t : G$.

Conversely, being given a pre-model $\mathcal{P} = (v', *', \models_p)$ we define $F \in *(t)$ iff $F \in *'(t)$ and $\mathcal{P} \models_p F$ for every polynomial t and every formula F. It is easy to see that $*$ is a proof-theorem assignment. Now we can define the model $\mathcal{M} = (v', *, \models)$ and prove that it is equivalent to the initial pre-model $\mathcal{P}$. As before we consider only formulae of the form $t : G$.

$$\mathcal{M} \models t : G \Leftrightarrow \quad G \in *(t) \quad \Leftrightarrow \quad G \in *'(t) \text{ and } \mathcal{P} \models_p G \quad \Leftrightarrow \quad \mathcal{P} \models_p t : G.$$

Reflexivity of $\mathcal{M}$ immediately follows from reflexivity of $\mathcal{P}$. The only thing we have to show is that $*$ is transitive in case of transitive $*'$. Suppose $F \in *(t)$. It means that $F \in *'(t)$ and $\mathcal{P} \models_p F$. Then $(t : F) \in *'(!\, t)$ since $*'$ is transitive and obviously $\mathcal{P} \models_p t : F$. So $(t : F) \in *(!\, t)$. □

Notion of a CS-pre-model is defined similarly to that of a CS-model (see Definition 4).

Definition 9. *A pre-model $\mathcal{P}$ is called*

- *an $\mathcal{LP}(\mathcal{T})$*-pre-model *if it is a $CS_{\mathcal{LP}(\mathcal{T})}$-pre-model;*
- *an $\mathcal{LP}(\mathcal{S}4)$*-pre-model *if it is a $CS_{\mathcal{LP}(\mathcal{S}4)}$-pre-model with a transitive proof-theorem assignment.*

By Theorem 3 and Lemma 3 we have

Theorem 4. *Let $L \in \{\mathcal{LP}(\mathcal{T}), \mathcal{LP}(\mathcal{S}4)\}$, then*

$$L \vdash F \quad \Leftrightarrow \quad F \text{ is true in all } L\text{-pre-models.}$$

3 The Decision Algorithm

In this section we describe the decision algorithm for non-derivability problem in explicit modal logics (this problem is dual to derivability problem) and evaluate its complexity. The decision procedure is based on Theorem 3 (or on Theorem 4 for reflexive logics $\mathcal{LP}(\mathcal{T})$, $\mathcal{LP}(\mathcal{S}4)$). Given a formula F in order to establish that $L \nvdash F$ one can construct an L-model $\mathcal{M}$ such that $\mathcal{M} \not\models F$ if L is one of the logics $\mathcal{LP}(\mathcal{K})$, $\mathcal{LP}(\mathcal{D})$, $\mathcal{LP}(\mathcal{K}4)$, $\mathcal{LP}(\mathcal{D}4)$ (or an L-pre-model $\mathcal{P}$ such that $\mathcal{P} \not\models_p F$ for $L \in \{\mathcal{LP}(\mathcal{T}), \mathcal{LP}(\mathcal{S}4)\}$). The algorithm consists of two parts.

1. *The saturation algorithm* produces a set of requirements which should be imposed on a counter-model for the formula F.
2. *The completion algorithm* constructs a counter-model satisfying these requirements if such a model exists.

Along with formulae we also consider expressions of the form $A \in *(t)$. We call these expressions $*$-requirements. Formulae and $*$-requirements are called metaformulae. A sequent is a pair $\Gamma \Rightarrow \Delta$, where Γ and Δ are finite sets of metaformulae.

Definition 10. *A sequent $\Gamma \Rightarrow \Delta$ is* true *in a model (pre-model) if at least one metaformula from Γ is false or at least one metaformula from Δ is true in it.*

Definition 11. *A sequent $\Gamma \Rightarrow \Delta$ is* saturated *if*

1. *$(A \to B) \in \Gamma$ implies $A \in \Delta$ or $B \in \Gamma$*
2. *$(A \to B) \in \Delta$ implies $A \in \Gamma$ and $B \in \Delta$*
3. *$(t : A) \in \Gamma$ implies $(A \in *(t)) \in \Gamma$*
4. *$(t : A) \in \Delta$ implies $(A \in *(t)) \in \Delta$*

A sequent $\Gamma \Rightarrow \Delta$ is reflexively saturated *if in the previous list we replace the conditions 3 and 4 by their reflexive analogues.*

3′. *$(t : A) \in \Gamma$ implies $A \in \Gamma$ and $(A \in *(t)) \in \Gamma$*
4′. *$(t : A) \in \Delta$ implies $A \in \Delta$ or $(A \in *(t)) \in \Delta$*

3.1 The Saturation Algorithm

In this subsection we describe *saturation algorithm* and evaluate its complexity. We describe the algorithm in details for the case of $\mathcal{LP}(\mathcal{S}4) = \mathcal{LP}$ and then point out the amendments that should be done to adapt the algorithm for other logics.

Algorithm starts being given a sequent $\Gamma \Rightarrow \Delta$. Every formula in it can be discharged (unavailable) or undischarged (available for processing). Initially all formulae are undischarged. Non-deterministically choose some undischarged formula G from $\Gamma \cup \Delta$ and non-deterministically try to perform one of the following instructions.

1. If $G \equiv (A \to B) \in \Gamma$ then put A into Δ or B into Γ
2. If $G \equiv (A \to B) \in \Delta$ then put A into Γ and B into Δ
3. If $G \equiv (t : A) \in \Gamma$ then put A and $(A \in *(t))$ into Γ
4. If $G \equiv (t : A) \in \Delta$ then put A or $(A \in *(t))$ into Δ

After a step is performed discharge G (make it unavailable). Discharge G even if it is a sentence letter or $\bot$ and none of the clauses above could be applied. Terminate if all formulae from $\Gamma \cup \Delta$ are discharged. Produce the obtained sequent as a result.

Lemma 4. *The saturation algorithm satisfies the following properties.*

1. *It terminates.*
2. *It produces a reflexively saturated sequent.*
3. *For every pre-model the initial sequent is false in it whenever the resulting one is false.*
4. *For every pre-model if the initial sequent is false in it then one of the possible computations produces a sequent, which is also false in it.*

Proof. 1. Let us define the depth of a formula by induction $d(S_i) = d(\bot) = 1$, $d(A \to B) = d(A) + d(B) + 1$, $d(t : A) = d(A) + 1$. Obviously, each step of the algorithm decreases the sum of the depths of all available formulae in the sequent. Therefore, the algorithm terminates.

2. Each step of the algorithm performs saturation for the chosen formula. Since all formulae in the resulting sequent are discharged this sequent is reflexively saturated.
3. It is easy to see from the definition of the saturation algorithm that if we reverse the algorithm step by step the falseness of the sequent preserves. So from the assumption that the resulting sequent is false we derive that the initial one is necessarily false.
4. Suppose the initial sequent is false in a given pre-model. All metaformulae are true or false in it. We start the algorithm. At every step we can put the metaformula to Γ if it is true and to Δ if it is false.

□

Corollary 1. *Given a formula F put $\Gamma := \emptyset$, $\Delta := \{F\}$. Perform the saturation algorithm for the sequent $\Gamma \Rightarrow \Delta$. If the saturation algorithm produces a sequent which is false in some $\mathcal{LP}(\mathcal{S}4)$-pre-model then $F \notin \mathcal{LP}(\mathcal{S}4)$. Otherwise, if every possible computation leads to an $\mathcal{LP}(\mathcal{S}4)$-valid sequent, i.e. a sequent true in all $\mathcal{LP}(\mathcal{S}4)$-pre-models, then $F \in \mathcal{LP}(\mathcal{S}4)$.*

Lemma 5. *The saturation algorithm is an NP-algorithm (Σ_1^p in the polynomial hierarchy), i. e. it is a non-deterministic algorithm that works polynomial time.*

Proof. The length of all branches of the computational tree is limited by the number of subformulae of the initial sequent. The number of variants of processing on every step of the algorithm is twice as large because some formulae can be processed in two different ways. We only need to find the branch of the computational tree that will produce a sequent that is not $\mathcal{LP}(\mathcal{S}4)$-valid. So the computational tree is a NP-tree. □

Now let us mention a useful property of the saturation algorithm.

Lemma 6. *If performing instructions of the saturation algorithm one would erase the discharged formula then Lemma 4 and Lemma 5 remain true.*

In what follows, we will use this second variant of the saturation algorithm.

Remark 4. Now we describe how to adapt the saturation algorithm for non-reflexive logics $\mathcal{LP}(\mathcal{K})$, $\mathcal{LP}(\mathcal{D})$, $\mathcal{LP}(\mathcal{K}4)$ and $\mathcal{LP}(\mathcal{D}4)$. Since we need to construct a model (not a reflexive pre-model as before) we do not need a reflexively saturated sequent and the instructions for processing $t : A$ should be read as follows:

$3'$. If $G \equiv (t : A) \in \Gamma$ then put $(A \in *(t))$ into Γ
$4'$. If $G \equiv (t : A) \in \Delta$ then put $(A \in *(t))$ into Δ

This saturation algorithm has the same properties except for one. It produces a saturated sequent (not a reflexively saturated one).

3.2 The Completion Algorithm

As before, first we discuss the completion algorithm for $\mathcal{LP}(\mathcal{S}4)$ and then adapt it for other explicit logics. The completion algorithm deals with the sequent $\Gamma \Rightarrow \Delta$ containing atomic formulae and $*$-requirements. It terminates with success if there exists an $\mathcal{LP}(\mathcal{S}4)$-pre-model in which $\Gamma \Rightarrow \Delta$ is false. Otherwise, it terminates with failure. Let us clarify when such a pre-model exists and how it should be constructed.

Of course, if $\Gamma \cap \Delta \neq \emptyset$ or $\bot \in \Gamma$ then the counter-model in question cannot exist. Indeed, $\bot$ is always false and no formula can be true and false simultaneously.

Suppose all of the assumptions above are wrong. Then we can define a truth-assignment $\overline{v}$ as follows

$$\overline{v}(S_i) = \mathit{True} \quad \Leftrightarrow \quad S_i \in \Gamma. \tag{2}$$

Then sentence letters from Γ are true and the letters from Δ are false. Thus, in order to construct a counter-model it is sufficient to satisfy the $*$-requirements including transitivity of it. Besides, the counter-model in question should be a $CS_{\mathcal{LP}(\mathcal{S}4)}$-pre-model which can also be expressed in terms of $*$-requirements. Let $CS^*_{\mathcal{LP}(\mathcal{S}4)}$ denotes the set

$$CS^*_{\mathcal{LP}(\mathcal{S}4)} = \{\mathbf{A} \in *(a) \,|\, a \text{ is a proof constant, } \mathbf{A} \text{ is an axiom of } \mathcal{LP}(\mathcal{S}4)\}.$$

Therefore, in order to construct the counter-model for $\Gamma \Rightarrow \Delta$ it is sufficient to produce a transitive proof-theorem assignment $*$ such that all $*$-requirements from Γ and $CS^*_{\mathcal{LP}(\mathcal{S}4)}$ are true and all $*$-requirements from Δ are false for $*$.

Definition 12. *Let Φ be an arbitrary set of $*$-requirements. A proof-theorem assignment $*$ is* based *on Φ if all requirements from Φ are true for $*$.*

Lemma 7. *For any set Φ there exists a minimal transitive proof-theorem assignment $*$ based on it, i. e. $*$ is based on Φ and for every transitive proof-theorem assignment $*'$ based on Φ we have $*(t) \subseteq *'(t)$ for all polynomials t.*

Proof. In order to construct such an assignment we should only close Φ under the following rules.

R1 $\dfrac{G \in *(t)}{t : G \in *(!\, t)}$

R2 $\dfrac{(A \to G) \in *(t) \quad A \in *(s)}{G \in *(t \cdot s)}$

R3 $\dfrac{G \in *(t_i)}{G \in *(t_1 + t_2)}, i = 1, 2$ □

Let Γ' and Δ' denote sets of $*$-requirements from Γ and Δ respectively.

Lemma 8. *Let $\Gamma \Rightarrow \Delta$ be a sequent containing only atomic formulae and $*$-requirements. It is refutable, i.e. there is an $\mathcal{LP}(\mathcal{S}4)$-pre-model that refutes it, iff the following conditions are satisfied.*

1. $\Gamma \cap \Delta = \emptyset$
2. $\bot \notin \Gamma$
3. *All $*$-requirements from Δ' are false for the minimal transitive proof-theorem assignment $*_m$ based on $\Gamma' \cup CS^*_{\mathcal{LP}(\mathcal{S}4)}$*

Proof. We consider the minimal transitive proof-theorem assignment $*_m$ based on $\Gamma' \cup CS^*_{\mathcal{LP}(\mathcal{S}4)}$. If this assignment refutes all $*$-requirements from Δ' then the pre-model $\mathcal{P} = (\overline{v}, *_m, \models_p)$ (see (2)) refutes $\Gamma \Rightarrow \Delta$. Otherwise, if $*_m$ satisfies one of the $*$-requirement from Δ' then it is true for any other transitive assignment based on $\Gamma' \cup CS^*_{\mathcal{LP}(\mathcal{S}4)}$. So the desired counter-model does not exist.

In order to deal with axiom schemes we add to the language of $\mathcal{LP}$ formula variables $T_1, \ldots, T_n, \ldots$ and polynomial variables $r_1, \ldots, r_n, \ldots$. It makes

possible writing one formula in the extended language instead of an infinite set of formulae in the language of $\mathcal{LP}$. Suppose we need to find an intersection of the schemes A and B, i. e. the set of $\mathcal{LP}$-formulae whose structure satisfies the scheme A together with the scheme B. An obvious way of solving this problem is to find the most general unifier (mgu) of A and B. This unification means that we substitute polynomial variables by some polynomials in the extended language and formula variables by some formulae in the extended language.

In what follows, by a formula we mean a formula in the extended language.

Now let us describe the *completion algorithm.*

Suppose that

$$\Gamma' = \{A_1 \in *(t_{i_1}), \ldots, A_n \in *(t_{i_n})\},$$
$$\Delta' = \{B_1 \in *(s_{j_1}), \ldots, B_m \in *(s_{j_m})\}.$$

Some of t_{i_k} and s_{j_l} can coincide.

Preliminary operations. Terminate with failure if $\Gamma \cap \Delta \neq \emptyset$ or $\bot \in \Gamma$.

Otherwise, non-deterministically choose one of s_{j_l}, $l = 1, 2, \ldots, m$, and perform the following actions with it.

Initialization. Non-deterministically choose several non-intersecting occurrences of t_{i_k}, $k = 1, 2, \ldots, n$, as subpolynomials of s_{j_l}. Let us call the chosen occurrences pseudo-elementary polynomials. Polynomial s_{j_l} is considered to be built from pseudo-elementary polynomials, proof variables and constants. To every chosen occurrence of t_{i_k} non-deterministically assign one of the formulae A such that $(A \in *(t_{i_k})) \in \Gamma'$. Non-deterministically assign to every occurrence of axiom constants (except those in pseudo-elementary polynomials) one of axiom schemes written as one formula in the extended language. Choose different formula and polynomial variables for different occurrences of axiom constants.

Assign to every occurrence of + (except those in pseudo-elementary polynomials) one of two symbols 'l' or 'r'. Assign *null* to the occurrences of proof variables that are not assigned yet. So assigning *null* to a subpolynomial actually means that nothing is assigned to this subpolynomial. Initialization is complete.

Assigning. Assign formulae to subpolynomials of s_{j_l} according to the following rules. Suppose formulae C_1 and C_2 are assigned to occurrences of subpolynomials q_1 and q_2 respectively. One of C_1 and C_2 or both of them may be *null.*

1. Assign the formula C_1 to $q_1 + q_2$ if 'l' was assigned to this occurrence of +. Otherwise, assign C_2.
2. Assign the formula $q_1 : C_1$ to $!\, q_1$ if C_1 is not *null.* Otherwise, assign *null.*
3. Assign *null* to $q_1 \cdot q_2$ if C_1 is neither a formula variable nor a formula of the form $D \to E$, or if C_2 is *null.* Otherwise, if the main connective in C_1 is implication then unify D and C_2. If the formulae are unifiable find their mgu σ and assign $E\sigma$ to $q_1 \cdot q_2$. If unification is impossible assign *null.* Finally, if C_1 is a formula variable T then assign some new formula variable T' to $q_1 \cdot q_2$.

Checking. Finally, some formula is assigned to the polynomial s_{j_l}. Unify it with the formula B_l. Terminate with failure if these formulae are unifiable. Otherwise, if unification is impossible or *null* is assigned to s_{j_l} perform another initialization and proceed as before.

If none of the initializational variants terminates with failure then choose another polynomial s_{j_l} and perform initializations for it. Terminate with success if processing none of the polynomials s_{j_l}, $l = 1, \dots, m$, terminates with failure.

Lemma 9. *Suppose a sequent $\Gamma \Rightarrow \Delta$ consists of atomic formulae and $*$-requirements. The completion algorithm terminates with success on this sequent iff the sequent $\Gamma \Rightarrow \Delta$ is not $\mathcal{LP}(\mathcal{S}4)$-valid.*

Now let us evaluate the complexity of the completion algorithm. In the process of its execution we need to perform multiple unifications. The length of the unified formulae may increase exponentially.

Example 1. Suppose Γ' contains the following $*$-requirements.

$$(T_1 \to (T_2 \to \dots \to (T_M \to \underbrace{T_1 \wedge T_1 \wedge \dots \wedge T_1}_{M}) \dots)) \in *(c_1),$$
$$(\underbrace{T_2 \wedge T_2 \wedge \dots \wedge T_2}_{M}) \in *(c_2),$$
$$\dots$$
$$(\underbrace{T_M \wedge T_M \wedge \dots \wedge T_M}_{M}) \in *(c_M).$$

Then we should assign

$$(T_2 \to \dots \to (T_M \to \underbrace{\underbrace{T_2 \wedge T_2 \wedge \dots \wedge T_2}_{M} \wedge \dots \wedge \underbrace{T_2 \wedge T_2 \wedge \dots \wedge T_2}_{M}}_{M}) \dots)$$

to $c_1 \cdot c_2$. All the initial requirements have the length $O(M)$ while this one is $O(M^2)$. Evidently, each step increases the length by M times. So the length of formula assigned to $c_1 \cdot c_2 \cdot \ldots \cdot c_M$ is $O(M^M)$.

In order to reduce complexity of the completion algorithm we can store formulae as direct acyclic graphs (dags). Then one can use the Robinson graph algorithm (for details cf. [4]) for unification of formulae that is polynomial of the sum of sizes of the dags. Using this algorithm for unification in the completion algorithm we obtain the following result.

Lemma 10. *The problem of realizing whether a given sequent containing only atomic formulae and $*$-requirements is refutable is a co-NP problem (Π_1^p in the polynomial hierarchy).*

Proof. It follows from complexity evaluation of completion algorithm since this algorithm solves the problem in question. □

Remark 5. For serial logics $\mathcal{LP}(\mathcal{D})$, $\mathcal{LP}(\mathcal{D}4)$ we need to check another trivial condition before we start constructing $*_m$: $(\bot \in *(t)) \notin \Gamma$ for all polynomials t.

In case this condition is not satisfied terminate with failure. For non-transitive logics $\mathcal{LP}(\mathcal{K})$, $\mathcal{LP}(\mathcal{D})$, $\mathcal{LP}(\mathcal{T})$ the set CS_L^* is defined as follows:

$CS_L^* = \{(!^{n-1}\, a{:}\ldots{:}a{:}\mathbf{A}) \in *(!^n\, a) \mid a \text{ is a proof constant, } \mathbf{A} \text{ is an axiom of } L\}$.

So instead of assigning axioms to occurrences of axiom constants during initialization we should non-deterministically assign formulae $!^{n-1}\, a : \ldots : a : \mathbf{A}$ to occurrences of polynomials $!^n\, a$. Also we should not use the rule **R1** for these logics.

Let us describe the *decision algorithm* for $\mathcal{LP}$. Given a formula F

1. Start the saturation algorithm on the sequent $\Rightarrow F$. It produces as a result the sequent $\Gamma \Rightarrow \Delta$.
2. Start the completion algorithm on the sequent $\Gamma \Rightarrow \Delta$.
3. Terminate with success if the completion algorithm terminates with success. Terminate with failure otherwise.

We summarize Corollary 1 and Lemma 9 in a theorem

Theorem 5. *Suppose L is an explicit logic. Given a formula F the decision algorithm terminates with success iff $F \notin L$.*

By Lemma 5 and Lemma 10 we have

Theorem 6. *The problem of L-satisfiability is Σ_2^p. Consequently, the problem of derivability in L is Π_2^p.*

Remark 6. Since all the logics under consideration are conservative extensions of the classical propositional logic the problem of L-satisfiability is NP-hard (Σ_1^p-hard).

Corollary 2. *The problem of L-satisfiability belongs to $\Sigma_2^p \cap \Sigma_1^p$-hard.*

References

1. Artemov, S.: Operational Modal Logic. Technical Report MSI 95–29. Cornell University (1995)
   <http://www.math.cornell.edu/~artemov/MSI95-29.ps>
2. Artemov, S.: Logic of Proofs: a Unified Semantics for Modality and λ-terms. Technical Report CFIS 98–06. Cornell University (1998)
   <http://www.math.cornell.edu/~artemov/CFIS98-06.ps>
3. Artemov, S.: Explicit Provability and Constructive Semantics. To appear in the Bulletin for Symbolic Logic
   <http://www.math.cornell.edu/~artemov/BSL.ps>
4. Bidoit, M., Corbin, J.: A Rehabilitation of Robinson's Unification Algorithm. Information Processing, Vol. 83. North-Holland (1983) 909–914
5. Brezhnev, V.: On Explicit Counterparts of Modal Logic. Manuscript (1999)
6. Gödel, K.: Eine Interpretation des intuitionistischen Aussagenkalkuls. Ergebnisse Math. Colloq., Bd. 4 (1933) 39–40
7. Mkrtychev, A.: Models for the Logic of Proofs. Lecture Notes in Computer Science, Vol. 1234. Springer-Verlag, Berlin Heidelberg New York (1997) 266–275

Finite Models and Full Completeness

J. Laird

COGS, University of Sussex
Brighton BN1 9QH, UK
jiml@cogs.susx.ac.uk

Abstract. A finite model property for *fully complete* denotational models of propositional logics is investigated using fully complete translations to compare programming languages and logics. The main result is that there can be no finite and fully complete models of linear or affine propositional logics. This is shown to be a consequence of Loader's result that contextual equivalence for finitary PCF is not decidable by giving a fully complete translation from finitary PCF into a $\Lambda(\Omega)^{\omega}bda$-calculus for a dual affine/non-linear logic. It is shown that the non-linear part of this logic does have a finite and fully complete model, and a conservative extension of the above translation is given from finitary PCF with control (μPCF) into the non-linear fragment which shows that the fully abstract model of μPCF is effectively presentable.

Keywords: linear logic, affine logic, full completeness, PCF, effective presentability.

1 Introduction

This paper concerns the possibility of a denotational semantics providing complete and effective information about proofs and programs. Completeness with respect to *proofs* rather than *provability* has been introduced as *full completeness* [3].

Definition 1. *A model of a logic $\mathcal{L}$ in (a category) $\mathcal{C}$ is* fully complete *if every morphism between objects $[\![A]\!]_{\mathcal{C}}$ and $[\![B]\!]_{\mathcal{C}}$ in $\mathcal{C}$ is the denotation of a proof of $A \vdash B$ in $\mathcal{L}$.*

Fully complete models will exist for all sufficiently well-behaved logics — 'term models' can be constructed from equivalence classes of the proofs themselves. But the main interest in the full completeness problem is in finding models which are 'syntax-independent'. As in the case of the full abstraction problem for PCF [24,4,14] it is not clear what syntax independence should mean. Is it sufficient to have a re-presentation of the syntax in a semantic form — or should denotational models have some inherently semantic quality?

A possible criterion is the following 'finite model property' for denotational models.

P. Clote and H. Schwichtenberg (Eds.): CSL 2000, LNCS 1862, pp. 384–398, 2000.

Definition 2. *A (categorical) model of proofs $\mathcal{C}$ is* finitary *if for each pair of formulas, A, B, there are finitely many elements $f_1, f_2, \ldots f_n \in \mathcal{C}([\![A]\!], [\![B]\!])$ and an effective procedure for listing them.*

The property of having *some* fully complete finite model is essentially a standard 'finite model property' for languages (such as classical propositional logic) with a traditional completeness theorem. To capture the finer detail of a denotational semantics, we shall say that a language possesses the 'finite *denotational* model property' only if it has a non-trivial finitary and fully complete models $\mathcal{C}$; i.e. there are some formulas such that $\mathcal{C}([\![A]\!]_{\mathcal{C}}, [\![B]\!]_{\mathcal{C}})$ contains more than on e morphism.

An effectiveness criterion has been incorporated (like the 'Jung and Stoughton condition' of effective presentability for fully abstract models of PCF [15]) — not only should the number of elements of the model be finite at each type but it should be possible to generate them in a finite time. Taken together, finiteness and full completeness mean having full information about each type-object of a model, making problems such as contextual equivalence for the language decidable.

Call a *logic* finitary, if every proposition has finitely many different proofs (modulo cut-elimination and commuting conversions). So intuitionistic logic, for instance, is *not* finitary as there are infinitely many 'different' cut-free proofs of $(P \Rightarrow P) \Rightarrow (P \Rightarrow P)$ — the Church numerals. 'Fully *and* faithfully complete models'(such as several games models [3,1,2]) which are by definition isomorphic to the term model will be finitary if and on ly if the logic is finitary.

Semantic models of propositional logics based on sets and functionals (possibly with additional structure) *will* be finitary provided the atomic propositions are represented as finite sets, and the the connectives as operations preserving finiteness. This is the case for the standard interpretation of the simply-typed λ-calculus or the coherence space models of propositional linear logic [10]. These models also tend to contain 'junk' — elements which are not the interpretations of proofs. Jung and Tiuryn [5] and others [12,28] have shown that it is possible to 'cut down' these models by constructing a definability predicate (in an apparently syntax independent way) to get a full completeness result. But can this be done effectively ? This is precisely the question posed by Streicher in [28] with respect to the coherence space model of linear logic in which the atoms are interpreted as finite cliques.

> Is it decidable whether a clique in the coherence space model comes from a proof?

In fact we can answer this question in the negative because there can be no (effective) finite models of propositional linear logic which are complete in the weaker traditional sense. This has been shown directly by Lafont [17] but is a simple corollary of the well-known result that provability for linear logic is not decidable.

Proposition 1. *If a logic has a finite and fully complete model, then it is decidable.*

Theorem 1 (Lincoln et al. [20]). *Propositional linear logic is undecidable.*

Corollary 1. *There are no finite and fully complete models of propositional linear logic.*

This paper will show that the converse to Proposition 1 is not true, however. Undecidability of propositional linear logic relies heavily on the restriction of weakening — *affine* propositional logic (which allows unrestricted weakening but retains the restrictions on contraction) *is* decidable, as shown by Kopylov [16]. The main result established here is that affine logic cannot, however, have any non-trivial finitary and fully complete models.

The basis for proving this is Loader's theorem that observational equivalence in finitary PCF is not decidable [22]. This is an important limitative result for denotational semantics, as it shows that there can be no effective presentation of a fully abstract model of PCF (and no finitary and fully complete model of finitary PCF). The analogous property of affine logic can be shown by demonstrating a tight connection between logic and programming language. Specifically, by giving a *fully complete translation* from finitary PCF into a fragment of linear logic. Because this translation is effective, a finitary and fully complete model would enable observational equivalence of PCF terms to be decided by comparing their translations. But this is not possible, so the re can be no effective, fully complete and finitary model of any part of linear logic containing the fragment in question. Thus the translation sheds some light on both Loader's result and decidability questions for linear logic.

Next, it is shown that an infinitary logic with a finitary and fully complete model does exist — intuitionistic propositional logic — using Padovani's effective characterization of the minimal model of the λ-calculus [26]. This result is used to show that the fully abstract model of finitary μPCF (PCF with control operators) *is* effectively presentable (by contrast to PCF), by giving a fully complete cps translation into the simply-typed λ-calculus with pairing whic h is a conservative extension of the translation from PCF.

In summary, the paper shows that the finite denotational model property is stronger than the simple finite model property, but not so strong that it precludes all infinitary logics. It does so by establishing a strong 'Curry-Howard style' correspondence with related results for simple λ-calculus based languages which allows effective presentability and definability to be analyzed from both perspectives.

2 A Linear/Non-linear λ-Calculus

The fragment of affine logic for which the translation will be given uses dual affine and intuitionistic contexts (and connectives) in a natural deduction presentation. (Similar to Benton's LNL calculus [7], Barber's DILL [6], and Girard's LU [11].) Formulas are generated from a single propositional atom ι by the connectives $\Rightarrow, \multimap, \times$ (intuitionistic implication, linear implication, and linear additive product). Thus it is expressive — it contains elements of multiplicative ($\multimap$), additive

($\times$) and exponential ($\Rightarrow$) affine logic. But it is also simple because the connectives which make difficulties for determining proof-equalities by giving rise to 'commuting conversions — ! and $\otimes$ — have been avoided. So the term-language is familiar — a λ-calculus with pairing — and the typing rules are based on a simple intuition — the linear implication can be introduced on ly when binding variables which occur linearly.

Definition 3. *Define the $\lambda_{L/NL}$-types (propositions) by induction over the following grammar:*

$$T ::= \iota \mid T \multimap T \mid T \times T \mid T \Rightarrow T$$

Terms-in-context of the associated calculus have the form $\Gamma; \Sigma \vdash t : T$, with a single typed formula on the right of the turnstyle, and two 'zones' of variables to the left. This allows some control over the use of structural rules without requiring explicit use of exponentials. The first zone is non-linear (Γ represents a set of variables), allowing contraction to take place there. The second zone is affine (Σ represents a multiset of variables) — contraction is not perm itted here.

The term-language is just the simply-typed λ-calculus with pairing. A single, standard notation for λ-abstraction is used, the distinction between the introduction rules for the two implication types being which zone the abstracted variable comes from. (The affine implication types are implicitly subtypes of non-linear implication types.) Similarly, a single (standard) notation for application is used for both linear and non-linear implication types. The only difference here between the elimination rules for the two versions of the implication is that a term of non-linear type can only be applied to terms containing no free linear variables, as non-linear application incorporates the 'promotion' rule of linear logic.

$$\frac{}{\_;x{:}T\vdash x{:}T}\ \text{Linear Axiom} \qquad \frac{\Gamma;\Sigma,x{:}S\vdash t{:}T}{\Gamma,x{:}S;\Sigma\vdash t{:}T}\ \text{Dereliction}$$

$$\frac{\Gamma;\Sigma\vdash t{:}T}{\Gamma;\Sigma,x{:}S\vdash t{:}T}\ \text{Weakening} \qquad \frac{\Gamma,x{:}S,y{:}S;\Sigma\vdash t{:}T}{\Gamma,z{:}s;\Sigma\vdash t[z/x,z/y]{:}T}\ \text{Contraction}$$

$$\frac{\Gamma;\Sigma,x{:}S\vdash t{:}T}{\Gamma;\Sigma\vdash \lambda x.t{:}S\multimap T}\ \multimap\text{-Intro} \qquad \frac{\Gamma;\Sigma\vdash s{:}S \quad \Gamma;\Sigma'\vdash t{:}S\multimap T}{\Gamma;\Sigma,\Sigma'\vdash t\ \ s{:}T}\ \multimap\text{-Elim}$$

$$\frac{\Gamma,x{:}S;\Sigma\vdash t{:}T}{\Gamma;\Sigma\vdash \lambda x.t{:}S\Rightarrow T}\ \Rightarrow\text{-Intro} \qquad \frac{\Gamma;\_\vdash s{:}S \quad \Gamma;\Sigma\vdash t{:}S\Rightarrow T}{\Gamma;\Sigma\vdash t\ \ s{:}T}\ \Rightarrow\text{-Elim}$$

$$\frac{\Gamma;\Sigma\vdash s{:}S \quad \Gamma\,\Sigma\vdash t{:}T}{\Gamma;\Sigma\vdash \langle s,t\rangle{:}S\times T}\ \times\text{-Intro} \qquad \frac{\Gamma;\Sigma\vdash t{:}T_1\times T_2}{\Gamma;\Sigma\vdash \pi_i(t){:}T_i}\ \times\text{-Elim}$$

Table 1. Term-formation rules for $\lambda_{L/NL}$

Definition 4. *The equational theory of $\lambda_{L/NL}$, $=_{\beta\eta\pi}$ is given by the reflexive, transitive closure of the following rules:*

$$(\beta) \quad (\lambda x.t)\ s =_{\beta\eta\pi} t[s/x]$$

$$(\eta) \quad \lambda x.(t\ x) =_{\beta\eta\pi} t,\ x \notin FV(t)$$

$$(\pi) \quad \pi_i(\langle t_1, t_2\rangle) =_{\beta\eta\pi} t_i,\ i = 1, 2$$

$$(\pi_\eta) \quad \langle \pi_1(t), \pi_2(t)\rangle = t$$

This paper will show that there are no finitary and fully complete models of $\lambda_{L/NL}$. Generalizing this result relies on the existence of *fully complete* translations into more standard logics.

Definition 5. *Let $\mathcal{L}_1, \mathcal{L}_2$ be logics (or typed languages).*
A translation $\phi : \mathcal{L}_1 \longrightarrow \mathcal{L}_2$ is fully complete if for every context Γ and type A, and for every derivation π in $\mathcal{L}_2$ of $\phi(\Gamma) \vdash \phi(A)$, there is a derivation λ of $\Gamma \vdash A$ in $\mathcal{L}_1$ such that $\phi(\lambda) = \pi$.

The existence of a fully complete translation $\phi : \mathcal{L}_1 \to \mathcal{L}_2$ means that a fully complete model of $\mathcal{L}_2$ gives rise to a fully complete model of $\mathcal{L}_1$, by taking $[\![\Gamma \vdash \lambda : A]\!] = [\![\phi(\Gamma) \vdash \phi(\lambda) : \phi(A)]\!]$.

Proposition 2. *If (intuitionistic) affine logic has the finite (denotational) model property then so does $\lambda_{L/NL}$.*

Proof. There is a fully complete translation from $\lambda_{L/NL}$into affine logic. This is in effect the well-known 'Girard translation' [10] from intuitionistic to linear logic, based on the decomposition of the implication $A \Rightarrow B$ as $!A \multimap B$. This is formally proved in [13] to be fully complete. The adaptation to the affine case is straightforward.

Kopylov's result [16] (affine logic is decidable) yields the following corollary.

Corollary 2. *For every type T there is an effective procedure for deciding whether some $t : T$ exists.*

There are further fully complete translations from affine logic into intuitionistic logic, and from intuitionistic into classical logic. Hence the failure of the finite denotational model property to hold for $\lambda_{L/NL}$implies that it does not hold for any of these logics, but as this is in any case a consequence of the failure of the weaker finite model property, these are left to a future paper.

3 Finitary PCF and Loader's Theorem

PCF is a simply-typed λ-calculus with constants. Finitary PCF has a single base type of booleans, with two values, **tt** and **ff**, a constant for non-termination Ω. and a conditional If ... then ... else ....

Definition 6. *Finitary* PCF*-types are given by the grammar:*

$$T ::= \mathtt{bool} \mid T \Rightarrow T$$

PCF *terms are given over contexts of typed variables by:*
$M ::= \Omega^T \mid \mathbf{tt} : \mathtt{bool} \mid \mathbf{ff} : \mathtt{bool} \mid x : T$
$(\text{If } M : \mathtt{bool} \text{ then } M : \mathtt{bool} \text{ else } M : \mathtt{bool}) : \mathtt{bool}$
$((M : S \Rightarrow T)\ M : S) : T \mid (\lambda(x : S).M : T) : S \Rightarrow T$
Define the equational theory of finitary PCF *as follows:*

$$(\lambda x.M)\ N =_{\mathrm{PCF}} M[N/x]$$

$$\text{If } \mathbf{tt} \text{ then } M \text{ else } N =_{\mathrm{PCF}} M$$

$$\text{If } \mathbf{ff} \text{ then } M \text{ else } N =_{\mathrm{PCF}} N$$

$$\text{If } \Omega \text{ then } M \text{ else } N =_{\mathrm{PCF}} \Omega$$

It is a well-known fact that every closed term of ground type is provably equivalent either to $\mathbf{tt}, \mathbf{ff}$ or Ω.

Definition 7 (Observational equivalence). *Given closed terms $M, N : T$, a compatible program context for M, N is a single-holed context $C[\cdot]$ such that $C[M], C[N]$ are closed terms of ground type.*
Then $M \simeq_T N$ (M is observationally equivalent to N) if and only if for all compatible program contexts $C[\cdot]$, $C[M] =_{\mathrm{PCF}} C[N]$.

Theorem 2. *[Loader [22]]The relation $\simeq_T$ is not decidable at all types.*

Finitary models of PCF certainly exist — for instance in the category of ordered sets and monotone functions. But a finitary and *and* fully complete model would allow contextual equivalence to be decided by generating bounded lists of terms containing representatives from each $\simeq$-equivalence class.

Definition 8. *A* listing algorithm *for a typed language is an effective procedure for generating a list of (closed) terms at each type T $M_1 : T, M_2 : T, \ldots M_{n_T} : T$ such that for all $N : T$ there exists $i \leq n$ such that $M_i \simeq N$.*

Proposition 3. *Given a listing algorithm for* PCF*, $\simeq$ is decidable.*

Proof. $M \simeq_T N$ if and only if for every context $C[\cdot]$, $\lambda x.C[x]\ M =_{\mathrm{PCF}} \lambda x.C[x]\ N$. Hence $M \simeq_T N$ if and only if for every $L : T \Rightarrow \mathtt{bool}$, $L\ M =_{\mathrm{PCF}} L\ N$. But if $L \simeq_{T \Rightarrow \mathtt{bool}} L'$, then $L\ M =_{\mathrm{PCF}} L'\ M$ and $L\ N =_{\mathrm{PCF}} L'\ N$. Hence if $L_1, L_2, \ldots, L_n$ is a list containing elements from every $\simeq_{T \Rightarrow \mathtt{bool}}$ equivalence class, $M \simeq_T N$ if and only if $L_i\ M =_{\mathrm{PCF}} L_i\ N$ for all N.

Proposition 4. *Given a finitary and fully complete model of* PCF*, there is a listing algorithm for* PCF*.*

Proof. At each type T there is a complete list of elements $e_1, e_2, \ldots e_{n_T} \in [\![T]\!]$. A list of terms $M_1, M_2, \ldots M_{n_T} : T$ such that $[\![M_i]\!] = e_i$ for all $i \leq n_T$ can be found because a recursive enumeration of *all* PCF-terms can be given, and searched for terms denoting each element (which will exist by full completeness). In any sound model, if $[\![M]\!] = [\![N]\!]$ then $M \simeq_T N$, so the list of terms $M_1, M_2, \ldots M_{n_T} : T$ contains elements from each contextual equivalence class.

Corollary 3. *There is no finite and fully complete model of finitary* PCF.

Unary PCF has a single base type unit containing two constants $\top, \bot$ (termination and non-termination) and a 'convergence test' for M, N : unit, If M then N, such that
If $\top$ then $M =_{\mathrm{PCF}} M$, If $\bot$ then $M =_{\mathrm{PCF}} \bot$.
Contextual equivalence can be defined as for finitary PCF, but as shown by Loader [23] and Schmidt-Schauß[27] (independently), it *can* be characterized effectively.

4 The Translation

The fully complete translation of PCF into $\lambda_{L/NL}$ is in essence very simple. It leaves the structure of PCF as a typed, call-by-name λ-calculus unchanged, translating the function type of PCF directly into the intuitionistic implication of $\lambda_{L/NL}$. The problem is to account for the constants of PCF: the type of booleans containing the *values* **tt** and **ff** and the non-termination constant Ω^T? The basic idea is to represent truth-values as 'church booleans' — left and right projections from the product $\iota \times \iota$ — in the linear function-space $\iota \times \iota \multimap \iota$.

Definition 9 ($\Box$-translation on PCF **types).**

$$\mathtt{bool}^{\Box} = \iota \times \iota \multimap \iota, \quad (S \Rightarrow T)^{\Box} = S^{\Box} \Rightarrow T^{\Box}$$

Non-termination is represented as the use of an undischarged non-linear assumption (i.e. a free variable) of 'empty type' ι.

Definition 10. *Let $x_\Omega : \iota$ be a unique $\lambda_{L/NL}$-variable, and $\{y^{\Box}, z^{\Box}, \ldots\}$ a set of $\lambda_{L/NL}$-variables distinct from x_Ω so that the correspondence with* PCF*-variables:*

$$y : T \;\longrightarrow\; y^{\Box} : T^{\Box}$$

is a bijection which extends to PCF *contexts.*
The $\Box$-translation is now defined as a mapping from PCF *terms-in-context to $\lambda_{L/NL}$-terms-in-context:*

$$\Gamma \vdash t \;\longrightarrow\; \Gamma^{\Box}, x_\Omega : \iota; \_ \vdash M^{\Box}$$

where $M^{\Box}$ is defined by structural induction as follows:

– $(z)^{\Box} = z^{\Box}$

- $(\Omega : \mathtt{bool})^\square = \lambda y : \iota \times \iota.(x_\Omega : \iota)$
- $(\Omega : S \Rightarrow T)^\square = \lambda y : S^\square.(\Omega : T)^\square$
- $(\lambda z.M)^\square = \lambda z^\square.M^\square$
- $(M\ N)^\square = M^\square\ N^\square$
- $\mathbf{tt}^\square = \lambda y : \iota \times \iota.\pi_1(y)$
- $\mathbf{ff}^\square = \lambda y : \iota \times \iota.\pi_2(y)$
- $(\text{If } L \text{ then } M \text{ else } N)^\square = \lambda y : \iota \times \iota.L^\square\ \langle M^\square\ y, N^\square\ y\rangle$

This is a sound definition — the translation of $\Gamma \vdash M$ is a derivable term-in-context of $\lambda_{L/NL}$ and it respects the operational rules of PCF.

Proposition 5. *The translation is sound with respect to* PCF *and $\beta\eta\pi$-equalities. i.e. if M, N are* PCF *terms and $M =_{\text{PCF}} N$, then $M^\square =_{\beta\eta\pi} N^\square$.*

Proof. It is straightforward to show that if $M =_\beta N$, then $M^\square =_\beta N^\square$. So it remains to observe that the conversions for the conditional are respected, for which a representative case is given:
$(\text{If } \mathbf{tt} \text{ then } M \text{ else } N)^\square = \lambda y.(\lambda x.\pi_1(x))\ \langle M^\square\ y, N^\square\ y\rangle$
$=_\beta \lambda y.\pi_1(\langle M^\square\ y, N^\square\ y\rangle) =_\pi \lambda y.M^\square\ y =_\eta M^\square$.

The translation of unary PCF is similar: $\text{let unit}^\square = \iota \multimap \iota$, and
$(\top)^\square = \lambda y.y$, $(\bot)^\square = \lambda y.x_\omega$, $(\text{If } M \text{ then } N)^\square = \lambda y.M^\square\ (N^\square\ y)$.

4.1 Normal Forms for $\boldsymbol{\lambda_{L/NL}}$

The key to proving fully completeness of the translation is a strong characterization of the $\beta\eta\pi$-equivalence classes of $\lambda_{L/NL}$ as *η-long normal forms*. With the exception of the typing restrictions, this is just the standard notion of η-long normal form for the simply-typed λ-calculus with pairing. They are defined here for a restricted set of types, sufficient to include the translations of PCF types.

Definition 11. *Define the* relevant *types by the following grammar:*

$$T ::= \iota \mid \iota \times \iota \mid \overline{T} \Rightarrow (T \multimap \iota)$$

where $\overline{R} \Rightarrow (S \multimap \iota)$ abbreviates $R_1 \Rightarrow (R_2 \Rightarrow (\ldots (R_n \Rightarrow (S \multimap \iota))\ldots))$.

Clearly the relevant types include all translations of PCF-types $T^\square$.

Definition 12 (η-long normal forms of $\lambda_{L/NL}$ over relevant types). *These are given as sets $N(\Gamma; \Sigma; T)$ of terms in context $\Gamma; \Sigma \vdash t : T$.*

$$\frac{y : \iota \in \Gamma \cup \Sigma}{y \in N(\Gamma; \Sigma; \iota)}$$

$$\frac{y : \iota \times \iota \in \Gamma \cup \Sigma}{\pi_1(y), \pi_2(y) \in N(\Gamma; \Sigma; \iota)}$$

$$\frac{r_i \in N(\Gamma; \_; R_i) i \leq n \quad s \in N(\Gamma; \Sigma; S) \quad x : \overline{R} \Rightarrow (S \multimap \iota) \in \Gamma}{x\ r_1 \ldots r_n\ s \in N(\Gamma; \Sigma; \iota)}$$

$$\frac{r_i \in N(\Gamma; \_; R_i) i \leq n \quad s \in N(\Gamma; \Sigma; S)}{x\ r_1 \ldots r_n\ s \in N(\Gamma; \Sigma, x : \overline{R} \Rightarrow (S \multimap \iota); \iota)}$$

$$\frac{s, t \in N(\Gamma; \Sigma; \iota)}{\langle s, t \rangle \in N(\Gamma; \iota \times \iota)}$$

$$\frac{t \in N(\Gamma, x_1 : R_1, \ldots, x_n : R_n; \Sigma, y : S; \iota)}{\lambda \overline{x}.\lambda y.t \in N(\Gamma; \Sigma; \overline{R} \Rightarrow (S \multimap \iota))}$$

It is necessary to establish that the η-long forms genuinely provide (unique) representatives of each $\beta\eta\pi$ equivalence classes.

Lemma 1. *Every term consisting of a single variable of relevant type $x : T$ is $\beta\eta\pi$-equivalent to a η-long form $\hat{x} : T$.*

Proof. Define $\hat{x}$ by induction on the type T:
If $T = \iota$, then $\hat{x} = x$,
If $T = \iota \times \iota$, then $\hat{x} = \langle \pi_1(x), \pi_2(x) \rangle$,
If $T = \overline{R} \Rightarrow (S \multimap \iota)$, then $\hat{x} = \lambda \overline{y}.\lambda z.x\ \overline{\hat{y}}\hat{z}$.

Proposition 6. *Let $\Gamma; \Sigma \vdash t : T$ be a term-in-context of $\lambda_{L/NL}$ at a translated* PCF *type. Then t is $\beta\eta\pi$-equivalent to a unique η-long normal form in $N(\Gamma; \Sigma; T)$.*

Proof. is by establishing the following property of $\lambda_{L/NL}$ terms, defined by induction on type-structure:
A term (with free variables) $t : \iota$ is *normalizable* if it is $\beta\eta\pi$ equivalent to a normal form t'.
A term $t : \iota \times \iota$ is normalizable if $\pi_1(t)$ and $\pi_2(t)$ are normalizable.
A term $t : \overline{S} \Rightarrow \overline{T} \multimap \iota$ is *normalizable* if for every sequence of normalizable terms of appropriate (and simpler) type, $s_1, \ldots, s_m$:
$t\ s_1 \ldots s_n : \iota$ is $\beta\eta\pi$ equivalent to a normal form.
Note that if t is normalizable, then t is itself $\beta\eta\pi$-equivalent to a normal form, as $t =_\eta \lambda \overline{x}.t\ \overline{x} =_{\beta\eta\pi} \lambda \overline{x}.t'$, where t' is a normal form of $t\ \overline{x}$.
Proof that all $\lambda_{L/NL}$-terms at relevant types are normalizable is by induction on the number of occurrences of application, pairing, or projection which they contain.
t is assumed to have the form $\lambda \overline{z}.r$, for some (possibly empty) sequence of variables $z_1, \ldots, z_n$ and term r which is *not* a λ-abstraction.

- If t contains no instances of application, then r is a variable x, where either $x \notin \{z_1, z_2, \ldots z_n\}$ or $x = z_j$ for some $j \leq n$.
  Then in case $x \neq z_j$ for any j, $t\ s_1 \ldots s_m =_\beta x\ s_{n+1} \ldots s_m$. By assumption $s_1, \ldots s_m$ are $\beta\eta\pi$ equivalent to normal forms $s'_1, \ldots s'_m$. So $t\ s_1 \ldots s_m$ is $\beta\eta\pi$-equivalent to the normal form $x\ s'_{n+1} \ldots s'_m$.
  In case $x = z_j$ for some $j \leq n$,
  $t\ s_1 \ldots s_m =_\beta s_j\ s_{n+1} \ldots s_m$, and by assumption of normalizability of s_j, $s_j\ s_{n+1} \ldots s_m$ is $\beta\eta\pi$ equivalent to a normal form.

– If $r = p\ q$ then if $s_1 \dots s_m$ are normalizable terms,
$t\ s_1 \dots s_m =_\beta ((\lambda \overline{z}.p)\ s_1 \dots s_n))\ (\lambda \overline{z}.q)\ s_1 \dots s_n)\ s_{n+1} \dots s_m$
By induction hypothesis, $\lambda \overline{z}.p, \lambda \overline{z}.q$ are normalizable, and hence $(\lambda \overline{z}.p)\ s_1 \dots s_n$ and $(\lambda \overline{z}.q)\ s_1 \dots s_n$ are normalizable.
Hence $((\lambda \overline{z}.p)\ s_1 \dots s_n))\ ((\lambda \overline{z}.q)\ s_1 \dots s_n)\ s_{n+1} \dots s_m$ is equivalent to a normal form as required.
– The cases $r = \pi_i(p)$, and $r = \langle p, q \rangle$ are similar (and simpler).

4.2 Completeness of the Translation

Proposition 7. *The $\square$-translation is fully complete.*

Proof. By Proposition 6, it is sufficient to show that for every η-long normal form $t \in N(\Gamma^\square, x_\Omega : \iota; \_; T^\square)$ there exists M such that $M^\square =_{\beta\eta\pi} t$.
This is shown by induction on the number of instances of application in t.
If $T = \texttt{bool}$ (the important case), then $t = \lambda y : \iota \times \iota.s$,
where $s \in N(\Gamma^\square, x_\Omega : \iota; y : \iota \times \iota; \iota)$

– If $s = x_\Omega$, then $t = \lambda y.x_\Omega = \Omega^\square$.
– If $s = \pi_1(y)$, or $\pi_2(y)$, then $t = \lambda y.\pi_1(y) = \mathbf{tt}^\square$ or $t = \lambda y.\pi_2(y) = \mathbf{ff}^\square$.
– If $s = z^\square\ r_1 \dots r_n$, then $z : S^\square$ for some PCF-type $S = \overline{R} \Rightarrow \texttt{bool} \in \Gamma$,
$r_1 \in N(\Gamma^\square, x_\Omega; \_; R_1^\square), \dots, r_{n-1} \in N(\Gamma^\square, x_\Omega; \_; R_1^\square)$, and so by hypothesis there are PCF terms $N_1, \dots N_{n-1}$ such that $N_i^\square =_{\beta\eta\pi} r_i$ for $i < n$.
$r_n : \iota \times \iota$, so by definition of $\lambda_{L/NL}$ normal forms, $r_n = \langle p_1, p_2 \rangle$ for some $p_1, p_2 \in N(\Gamma, x_\Omega; y : \iota \times \iota; \iota)$.
So $\lambda y.p_1, \lambda y.p_2 \in N(\Gamma^\square, x_\Omega; \_; \texttt{bool}^\square)$ are η-long normal forms at PCF-translated types, containing fewer applications than t. Hence by induction hypothesis there are PCF terms L_1, L_2 such that $L_i^\square =_{\beta\eta\pi} \lambda y.p_i$.
Putting these terms together, there is a PCF term:
$M =$ If $z\ N_1 \dots N_{n-1}$ then L_1 else L_2 such that
$M^\square = \lambda y.z^\square\ N_1^\square \dots N_{n-1}^\square\ \langle L_1^\square\ y, L_2^\square\ y \rangle =_{\beta\eta\pi} \lambda y.s = t$

If $T = R \Rightarrow S$, then $t = \lambda(z : R)^\square.(s : S^\square)$, where $s \in N((\Gamma, z)^\square, x_\Omega; \_; S^\square)$. By hypothesis, there is a PCF term $\Gamma, z : R \vdash N : S$ such that $N^\square = s$, so $(\lambda z.N)^\square = t$ as required.

Corollary 4. *If there is a fully complete and finitary model of $\lambda_{L/NL}$, then there is a fully complete and finitary model of* PCF.

Proof. Let $[\![\Gamma \vdash M : T]\!] = [\![\Gamma^\square, x_\Omega \vdash M^\square : T^\square]\!]$

Theorem 3. *There is no finite and fully complete model of $\lambda_{L/NL}$.*

Proof. Suppose a finitary and fully complete model of $\lambda_{L/NL}$exists. Then by Proposition 3 and Corollary 4, PCF observational equivalence is decidable. But this is a contradiction of Loader's result, Theorem 2.

Corollary 5. *There are no finitary and fully complete models of affine propositional logic.*

By contrast, effective presentability of the fully abstract model of unary PCF means that the corresponding fragment of linear logic has a finite model. The key difference between the translations appears to be the need to use the additive product to translate finitary PCF, whilst the translation of the unary version stays within the multiplicative fragment.

5 Finitary Full Completeness and Intuitionistic Logic

It has now been established that a logic exists (affine logic) which possesses the the standard finite model property, but not the finite model property for fully complete models. It is a natural question to ask whether there are any natural and non-trivial examples of non-finitary logics with the latter property.

The purely intuitionistic fragment of $\lambda_{L/NL}$ (with the connectives $\Rightarrow, \times$) is not finitary — it contains the Church numerals. It has been shown by Loader [21] that the problem of deciding definability in the standard sets-and-functions model of the λ-calculus is undecidable, so it is not possible to cut down this model effectively using a definability predicate as defined in [5].

However, finitary and fully complete models of this fragment do exist. The natural way to demonstrate this is by exhibiting such a model, and it is the object of a forthcoming paper is to do so, using the category of sequential algorithms [8]. However, a simple proof that there are finite and fully complete models of intuitionistis logic can be given as a corollary of Padovani's proof that the 'minimal model' of the λ-calculus *with constants* is effectively presentable. (For simplicity's sake, only the implicational fragment is considered, but the extension to products is straightforward.)

Theorem 4 (Padovani [26]). *Let $\Lambda(\top, \bot)$ be the simply-typed λ-calculus over the ground type ι with two ground-type constants, $\top, \bot : \iota$. Then the contextual equivalence on closed terms s, t of the same type:*
$s \simeq_{\top,\bot} t$ iff for all compatible closed closed contexts $C[\cdot] : \iota$, $C[s] =_\beta C[t]$
is decidable, and the 'minimal model' of $\simeq_{\top,\bot}$-classes of terms is effectively presentable.

The pure calculus has no ground type constants, and so contextual equivalence is tested at the type $\iota \Rightarrow (\iota \Rightarrow \iota)$.

Definition 13. *Contextual equivalence (the maximal non-trivial congruence containing $\beta\eta$-equality) on the simply-typed λ-calculus is defined as follows, for terms $s, t : T$*

$$s \simeq t \iff \forall(\text{closed}) C[\cdot] : \iota \Rightarrow (\iota \Rightarrow \iota),\ C[s] =_\beta C[t]$$

The minimal model *of the pure λ-calculus is the model in which $[\![s]\!] = [\![t]\!]$ if and only if $s \simeq t$.*

Proposition 8. *For all pure λ-terms $s, t : T$ (which are also terms of $\Lambda(\top, \bot)$)*

$$s \simeq t \quad \Longleftrightarrow \quad s \simeq_{\top,\bot} t$$

Proof. Suppose $s \not\simeq t$, then for some context $C^T[\cdot] : \iota \Rightarrow \iota \Rightarrow \iota$, $C[s] =_\beta \lambda xy.x$ and $C[t] = \lambda xy.y$, hence $C[s]\ \top\ \bot =_\beta \top$ and $C[t]\ \top\ \bot = \bot$, so $s \not\simeq_{\top,\bot} t$ as required.
Suppose $s \not\simeq_{\top,\bot} t$, then for some ground-typed $\Lambda(\Omega, \top)$ context $C^T[\cdot]$, $C[s] = \top$ and $C[t] = \bot$. Let x, y be variables not appearing in $C[\cdot]$, then $\lambda xy.C[s][x/\top][y/\bot] =_\beta \lambda xy.x$ and $\lambda xy.C[t][x/\top][y/\bot] =_\beta \lambda xy.y$ and so $s \not\simeq t$ as required.

The equivalence $\simeq_{\top,\bot}$ is extensional ($t \simeq^{S \Rightarrow T}_{\top,\bot} s$ if and only if $s\ r \simeq^T_{\top,\bot} t\ r$ for all $r : S$) and hence there are only finitely many $\simeq_{\top,\bot}$-equivalence classes at each type.

Corollary 6. *The minimal model of the pure simply-typed λ-calculus is finitary — i.e. there are finitely many $\simeq$-equivalenc classes at each type.*

To give a listing algorithm for the pure calculus, it is necessary to be able to determine which $\Lambda(\top, \bot)$-terms are equivalent to constant-free terms.

Definition 14. *A term $t : T$ of $\Lambda(\top, \bot)$ is* total *if $t[\bot/\top] \simeq_{\top,\bot} t[\top/\bot] \simeq_{\top,\bot} t$.*

Say that a type (i.e. proposition) T is *provable* if there is a closed term $t : T$ of the pure λ-calculus.

Proposition 9 (Weak Completeness). *T is a provable type if and only if there exists a total $\Lambda(\top, \bot)$ term of type T.*

Proof. All pure λ-terms are clearly total, so the implication from right to left is trivial. For the converse, a trivial induction suffices to show that any type T is provable if and only if $T \Rightarrow \iota$ is *not provable*. So suppose for a contradiction that there exists a total term $t : T$, where T is not provable, and hence there exists a pure λ-term $s : T \Rightarrow \iota$. Then $s\ t : \iota$ is a total term, but (up to β-equality) the only closed terms of type ι are $\top$ and $\bot$ which are obviously not total. This is the required contradiction.

Proposition 10 (Strong completeness). *A (closed) term $t : T$ of $\Lambda(\Omega)$ is total if and only if there exists a pure λ-term $s : T$ such that $s \simeq_{\top,\bot} t$.*

Proof. Suppose $t : \overline{S} \Rightarrow \iota$ is total. Then by Proposition 9 T is provable; there exists a pure λ-term $r : T$. Then there is a pure term $s = \lambda \overline{x} : \overline{S}.(t[r\ \overline{x}/\top][r\ \overline{x}/\bot])\ \overline{x}$ such that $s \simeq_{\top,\bot} t$. For any $\Lambda(\top, \bot)$ terms $p_1 : S_1, \ldots, p_n : S_n$, suppose w.l.o.g. $r\ p_1 \ldots p_n =_\beta \top$. Then $s\ p_1 \ldots p_n =_\beta t[\top/\bot]\ p_1 \ldots p_n =_\beta t\ p_1 \ldots p_n$ by definition of totality, and hence $s \simeq_{\top,\bot} t$.

Corollary 7. *The minimal model of the pure simply-typed λ-calculus is effectively presentable. (And so this is a finitary and fully complete model of intuitionistic implicational logic).*

Proof. By Padovani's result, there is a listing algorithm for the minimal model of $\Lambda(\top, \bot)$, and this yields a listing algorithm for the minimal model of the pure simply-typed λ-calculus since there is an effective procedure for finding the total terms of $\Lambda(\top, \bot)$ and for finding pure terms which are equivalent to them.

Compare the existence of a listing algorithm for this notion of minimal model with the undecidability of definability for set-theoretic models of the pure λ-calculus [21]. A similar contrast to Loader's result for finitary PCF is provided by the effectively presentable model of PCF with *control operators* described by Cartwright, Curien and Felleisen [9]. In fact, it will now be shown there is a continuation-passing style translation from such a language (finitary μPCF) into the *non-linear* fragment $(\Rightarrow, \times)$ of $\lambda_{L/NL}$ which is a *conservative extension* of the $\Box$-translation (on terms). Hence, the effective presentability of the minimal model can be used to give an alternative proof of effective presentability for the fully abstract model of finitary μPCF.

The language μPCF [25] is PCF extended with with first class continuations in the form of control operations called *naming* and *μ-abstraction*. Finitary μPCF has base types of booleans $\mathtt{bool}$ and the empty type $\mathbf{0}$. Terms are given in contexts (sets of typed variables, and *names* of ground type). (This is equivalent under call-by-name to adding names at all types [18]). The typing judgements for PCF are extended with the following rules:

$$\frac{\Gamma \vdash M : \mathtt{bool} ; \Delta}{\Gamma \vdash [\alpha]M : \mathbf{0} ; \Delta, \alpha} \qquad \frac{\Gamma \vdash M : \mathbf{0} ; \Delta, \alpha}{\Gamma \vdash \mu\alpha.M : \mathtt{bool} ; \Delta}$$

The equational theory of μPCF extends the PCF theory:

$$(\mu\alpha) \quad \mu\alpha.[\alpha]M =_{\mu PCF} M$$

and if $E[\cdot] = [\beta][\cdot]$, or $E[\cdot] = [\beta]\text{If } [\cdot] \text{ then } L \text{ else } N$ for some L, N,

$$(\mu\beta) \quad E[\mu\alpha.M] =_{\mu PCF} M[E[\cdot]/\alpha]$$

where $M[E[\cdot]/\alpha]$ means replace every named subterm $[\alpha]N$ in M with $E[N]$.

Definition 15 ($\Diamond$ translation). *Translation of types:*
$\mathbf{0}^\Diamond = \iota$, $\mathtt{bool}^\Diamond = (\iota \times \iota \Rightarrow \iota)$, *and* $(S \Rightarrow T)^\Diamond = S^\Diamond \Rightarrow T^\Diamond$.
The translation of terms *is conservative over the $\Box$-translation for PCF.*
For each name α, assume a distinguished variable $x_\alpha : \iota \times \iota$, and let $\{\alpha, \beta, \gamma, \ldots\}^\Diamond = \{x_\alpha : \iota \times \iota, x_\beta : \iota \times \iota, x_\gamma : \iota \times \iota\}$.
$\Gamma \vdash M : T ; \Delta$ is translated to $\Gamma^\Diamond, x_\omega : \iota, \Delta^\Diamond \vdash M^\Diamond : T^\Diamond$ as follows:

- $\Omega^\Diamond = \lambda y.x_\omega$, $\mathbf{tt}^\Diamond = \lambda x.\pi_1(x)$, $\mathbf{ff}^\Diamond = \lambda x.\pi_2(x)$
- $(\text{If } M \text{ then } N_1 \text{ else } N_2)^\Diamond = \lambda y.M^\Diamond \ \langle N_1^\Diamond \ y, N_2^\Diamond \ y\rangle$
- $(x)^\Diamond = x^\Diamond$, $(\lambda x.M)^\Diamond = \lambda x^\Diamond.M^\Diamond$, $(M \ N)^\Diamond = M^\Diamond \ N^\Diamond$
- $([\alpha]M)^\Diamond = (M^\Diamond \ x_\alpha)$, $(\mu\alpha.M)^\Diamond = \lambda x_\alpha.M^\Diamond$.

The proof of the following proposition follows that for Proposition 7, using η-normal forms.

Proposition 11. *The $\Diamond$-translation is fully complete.*

Corollary 8. *The $\Diamond$-translation is fully abstract:*
i.e. for all μPCF terms $M, N : T$ $M \simeq_T N$ if and only if $M^\Diamond \simeq N^\Diamond$.

Proof. Suppose $M^\Diamond \not\simeq N^\Diamond$. Then $C[M^\Diamond] =_\beta \lambda xy.x$ and $C[N^\Diamond] = \lambda xy.y$ for some context $C[\cdot] : \iota \Rightarrow \iota \Rightarrow \iota$. Then by full completeness of the translation there exists a μPCF term $L : T \Rightarrow \texttt{bool}$ such that $L^\Diamond = \lambda y : \iota \times \iota.(C[y]\ \pi_1(y)\ x_\omega)$. Then $(L\ M)^\Diamond = L^\Diamond\ M^\Diamond = tt^\Diamond$ and $(L\ N)^\Diamond = L^\Diamond\ N^\Diamond = \Omega^\Diamond$ and $M \not\simeq N$ as required.

Corollary 9. *The fully abstract model of finitary μPCF is effectively presentable.*

5.1 Further Work

Intuitionistic implicational logic has the finite denotational model property, whilst linear and affine logic do not. The most obvious open question, therefore (as for decidability of provability) concerns multiplicative-exponential logic. A solution to this problem can be expected to be hard — a finite and fully complete model of MELL would allow provability for this fragment to be decided, which itself is a difficult open problem [19]. Even finding a fully complete and finitary model of the multiplicative fragment of $\lambda_{L/NL}$ $(\multimap, \Rightarrow)$ — or a proof that no such model exists — is a goal which could provide insight into the decidability question.

Acknowledgements

The work reported here was undertaken at the LFCS in Edinburgh, with the support of the UK EPSRC grant "Foundational Structures in Computer Science". I would like to thank Samson Abramsky and Patrick Baillot for useful discussions.

References

1. S. Abramsky. Axioms for full abstraction and full completeness. In *Essays in Honour of Robin Milner*. MIT Press, to appear.
2. S. Abramsky and P.-A. Mellies. Concurrent games and full completeness. In *Proceedings of the 14th annual Symposium on Logic In Computer Science, LICS '99*, 1999.
3. S. Abramsky, R. Jagadeesan. Games and full completeness for multiplicative linear logic. *Journal of Symbolic Logic*, 59:543–574, 1994.
4. S. Abramsky, R. Jagadeesan and P. Malacaria. Full abstraction for PCF. Accepted for publication in Information and Computation, 1996.
5. A. Jung and J. Tiuryn. A new characterization of lambda definability. In *Typed Lambda Calculi and Applications*, number 664 in Lecture notes in Computer Science. Springer–Verlag, 1993.

6. A. Barber. Dual intuitionistic linear logic. Technical Report ECS-LFCS-96-347, LFCS, University of Edinburgh, 1996.
7. P.N. Benton. A mixed linear and non-linear logic: Proofs, terms and models. Technical Report 352, Computer laboratory, Cambridge University, 1994.
8. G. Berry and P.-L. Curien. Sequential algorithms on concrete data structures. *Theoretical Computer Science*, 20:265–321, 1982.
9. R. Cartwright, P.-L. Curien and M. Felleisen. Fully abstract semantics for observably sequential languages. *Information and Computation*, 1994.
10. J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50, 1987.
11. J.-Y. Girard. On the unity of logic. *Annals of Pure and Applied Logic*, 59:201–217, 1993.
12. J.-Y. Girard. On denotational completeness. *Theoretical Computer Science*, 197, 1995.
13. M. Hasegawa. Logical predicates for intuitionistic linear type theories. In *Proceedings of TLCA '99*, 1999.
14. J. M. E. Hyland and C.-H. L. Ong. On full abstraction for PCF: I, II and III. to appear, 1995.
15. A. Jung, and A. Stoughton. Studying the fully abstract model of PCF within its continuous function model. In *Proceedings of the International Conference on Typed Lambda Calculi and Applications*, volume 664 of *LNCS*, 1993.
16. A. Kopylov. Decidability of linear affine logic. In *Proceedings of the 10th Annual IEEE Symposium on Logic in Computer Science, LICS 95*. IEEE press, 1995.
17. Y. Lafont. The finite model property for various fragments of linear logic. *Journal of Symbolic logic*, 62:1202 — 1208, 1996.
18. J. Laird. *A Semantic Analysis of Control*. PhD thesis, Department of Computer Science, University of Edinburgh, 1998.
19. P. Lincoln. *Deciding provability of linear logic formulas*. Cambridge University Press, 1995.
20. P. Lincoln, J. Mitchell, A. Scedrov and N. Shankar. Decision problems for propositional linear logic. *Annals of Pure and Applied Logic*, 56:239–311, 1992.
21. R. Loader. The undecidability of lambda definability. to appear in the Church Memorial volume, 1993.
22. R. Loader. Finitary PCF is undecidable. Manuscript, 1997. To appear in Theoretical Computer Science.
23. R. Loader. Unary PCF is undecidable. *Theoretical Computer Science*, 206, 1998.
24. R. Milner. Fully abstract models of typed lambda-calculi. *Theoretical Computer Science*, 4:1–22, 1977.
25. C.-H. L. Ong and C. Stewart. A Curry-Howard foundation for functional computation with control. In *Proceedings of ACM SIGPLAN-SIGACT syposium on Principles of Programming Languages, Paris, January 1997*. ACM press, 1997.
26. V. Padovani. Decidability of all minimal models. In M. Coppo and S. Berardi, editor, *Types for proofs and programs*, volume 1158 of *LNCS*. Springer, 1996.
27. M. Schmidt-Schauß. Decidability of behavioural equivalence in unary PCF. *Theoretical Computer Science*, 208, 1998.
28. T. Streicher. Denotational completeness revisited. In *Proceedings of the International Conference on Categories in Computer Science, CTCS '99*, Electronic notes in theoretical computer science. Elsevier, 1999.

On the Complexity of Combinatorial and Metafinite Generating Functions of Graph Properties in the Computational Model of Blum, Shub and Smale

J.A. Makowsky[1,2,⋆] and K. Meer[3]

[1] Department of Computer Science
Technion–Israel Institute of Technology
Haifa, Israel
janos@cs.technion.ac.il

[2] Department of Mathematics
Swiss Federal Institute of Technology
Zurich, Switzerland

[3] Department of Mathematics and Computer Science
Syddansk Universitet, Odense, Denmark
meer@imada.sdu.dk

Abstract. We present a unified framework for the study of the complexity of counting functions and multivariate polynomials such as the permanent and the hamiltonian in the computational model of Blum, Shub and Smale. For $\mathbf{P}_{\mathbb{R}}$ we introduce complexity classes $Gen\mathbf{P}_{\mathbb{R}}$ and $CGen\mathbf{P}_{\mathbb{R}}$. The class $Gen\mathbf{P}_{\mathbb{R}}$ consists of the generating functions for graph properties (decidable in polynomial time) first studied in the context of Valiant's **VNP** by Bürgisser. $CGen\mathbf{P}_{\mathbb{R}}$ is an extension of $Gen\mathbf{P}_{\mathbb{R}}$ where the graph properties may be subject to numeric constraints.
We show that $Gen\mathbf{P}_{\mathbb{R}} \subseteq CGen\mathbf{P}_{\mathbb{R}} \subseteq EXPT_{\mathbb{R}}$ and exhibit complete problems for each of these classes. In particular, for $(n \times n)$ matrices M over $\mathbb{R}$, $ham(M)$ is complete for $Gen\mathbf{P}_{\mathbb{R}}$, but the exact complexity of $per(M) \in Gen\mathbf{P}_{\mathbb{R}}$ remains open. Complete problems for $CGen\mathbf{P}_{\mathbb{R}}$ are obtained by converting optimization problems which are hard to approximate, as studied by Zuckerman, into corresponding generating functions.
Finally, we enlarge once more the class of generating functions by allowing additionally a kind of non-combinatorial counting. This results in a function class $Met\text{-}Gen\mathbf{P}_{\mathbb{R}}$ for which we also give a complete member: evaluating a polynomial in the zeros of another one and summing up the results. The class $Met\text{-}Gen\mathbf{P}_{\mathbb{R}}$ is also a generalization of $\sharp\mathbf{P}_{\mathbb{R}}$, introduced by Meer, [Mee97].
Due to lack of space we will prove here only the $Met\text{-}Gen\mathbf{P}_{\mathbb{R}}$ result. In the full paper also the other theorems will be established rigorously.

⋆ partially supported by by the Fund for Promotion of Research of the Technion–Israeli Institute of Technology

P. Clote and H. Schwichtenberg (Eds.): CSL 2000, LNCS 1862, pp. 399–410, 2000.

1 Introduction

Valiant, in a fundamental paper [Val79], has shown that the matrix functions $per(M)$ and $ham(M)$ defined on $(n \times n)$ matrices $M = (m_{i,j})$ over a field $\mathbb{F}$ by

$$per(M) = \sum_{\pi \in \mathcal{S}_n} \prod_{i=1}^{n} m_{i,\pi(i)}$$

and

$$ham(M) = \sum_{\pi \in \mathcal{C}_n} \prod_{i=1}^{n} m_{i,\pi(i)}$$

are typically hard compute. Here $\mathcal{S}_n$ ($\mathcal{C}_n$) denotes the set of (cyclical) permutations of $\{1, ..., n\}$. To make the difficulty of computation precise, Valiant introduced his (non-uniform) model of computation of straight line programs and the complexity classes $\mathbf{VP}_{\mathbb{F}}$ and $\mathbf{VNP}_{\mathbb{F}}$, cf. [BCS97]. In spite of the beauty of this approach, there are various drawbacks in this model of computation, which the Blum, Shub and Smale model (BSS) of computation, cf. [BCSS98], proposes to overcome. In his paper [Mee97], Meer introduces counting problems over the reals $\mathbb{R}$ in the BSS model and its associated complexity class $\sharp\mathbf{P}_{\mathbb{R}}$. However, $per(M) \in \sharp\mathbf{P}_{\mathbb{R}}$ only for $m_{i,j} \in \{0, 1\}$. Meer analyses the logical definability of problems in $\sharp\mathbf{P}_{\mathbb{R}}$ and obtains results analogue to those of Saluja, Subrahmanyam and Thakur [SST95]. The issue of complete problems for $\sharp\mathbf{P}_{\mathbb{R}}$, however, is not discussed explicitely.

In this paper we work in the BSS-model and its extension to allow structured inputs as proposed by Grädel and Gurevich [GG98]. We propose a larger framework of counting functions based on an idea of Bürgisser [Bür99]. With the complexity class $\mathbf{P}_{\mathbb{R}}$ we associate new complexity classes $Gen\mathbf{P}_{\mathbb{R}}$, $CGen\mathbf{P}_{\mathbb{R}}$ and $Min\mathbf{P}_{\mathbb{R}}$ as classes of generating and maximizing functions of graph properties verifyable in $\mathbf{P}_{\mathbb{R}}$. Roughly speaking a function f in $Gen\mathbf{P}_{\mathbb{R}}$ has the following form: consider a meta-finite structure $\mathcal{D}$; its underlying finite structure should be $\langle A, R^A \rangle$ with finite universe $A = \{1, ..., n\}$ and relations R^A. Its weight functions are denoted by $\mathcal{W}$. Now let $\mathcal{E}$ be a $\mathbf{P}_{\mathbb{R}}$ computable class of meta-finite structures of signature $R, \mathcal{W}$ and additionally U. Here , U denotes relation symbols on the underlying finite universe. For the class $Gen\mathbf{P}_{\mathbb{R}}$ we consider membership of a meta-finite structure $\mathcal{A} := \langle A, R^A, U^A, \mathcal{W} \rangle$ in $\mathcal{E}$ as only depending on the underlying finite structure $\langle A, R^A, U^A \rangle$. An $f \in Gen\mathbf{P}_{\mathbb{R}}$ now is evaluated on $\mathcal{D}$; it depends on the R^A part of $\mathcal{A}$ and on an $\mathbb{R}$-term $t(\bar{x})$, which itself depends on the real number part $\mathcal{W}$ of $\mathcal{A}$; f has the form

$$Gen_{\mathcal{E}}(\langle A, R^A \rangle, t) = \sum_{\langle A, R^A, U \rangle \in \mathcal{E}} \prod_{\bar{x} \in U} t(\bar{x})$$

If the class $\mathcal{E}$ may also depend on the $\mathcal{W}$ part of $\mathcal{A}$ we get $CGen\mathbf{P}_{\mathbb{R}}$.
If instead of the sum of products we consider

$$Min_{\mathcal{E}}(\langle A, R^A \rangle, t) = \min_{\langle A, R^A, U \rangle \in \mathcal{E}} \sum_{\bar{x} \in U} t(\bar{x})$$

we have functions in $Min\mathbf{P}_{\mathbb{R}}$ and $CMin\mathbf{P}_{\mathbb{R}}$ respectively, which cover wide classes of Linear Programming problems with possible additional (non-linear) constraints.

To express, say, $ham(M)$ in this form, we take $\mathcal{E}$ to consist of structures $\langle A, R^A, U^A, w\rangle$ with $\langle A, R^A, w\rangle$ graphs with edge weights $w(x, y)$ in $\mathbb{R}$ and U^A ranges over all hamiltonian cycles of the graph and $t(x, y) = w(x, y)$. M is the weighted adjacency matrix of $\langle A, R^A\rangle$. If U^A ranges over all cycle covers rather than hamiltonian cycles, we get the permanent. It follows from the definitions that

$$Gen\mathbf{P}_{\mathbb{R}} \subseteq CGen\mathbf{P}_{\mathbb{R}} \subseteq EXPT_{\mathbb{R}}$$

and both $per(M) \in Gen\mathbf{P}_{\mathbb{R}}$ and $ham(M) \in Gen\mathbf{P}_{\mathbb{R}}$.

A typical example of a function in $Min\mathbf{P}_{\mathbb{R}}$ is given by the cost version of the Traveling Salesmen Problem TSP_{cost}. Given an edge weighted graph, compute the minimum cost of a hamiltonian tour. Here the cost is given by $t(\bar{x})$. Clearly we have

$$Min\mathbf{P}_{\mathbb{R}} \subseteq CMin\mathbf{P}_{\mathbb{R}} \subseteq EXPT_{\mathbb{R}}$$

but the relationship between $Min\mathbf{P}_{\mathbb{R}}$ and $Gen\mathbf{P}_{\mathbb{R}}$ (aa well as $CMin\mathbf{P}_{\mathbb{R}}$ and $CGen\mathbf{P}_{\mathbb{R}}$) remains, for the time being, open.

Now let $\gamma(U)$ be a polynomial time computable function of the weights of elements in U. We are thinking here of an additional constraint function. We modify the permanent and the hamiltonian matrix function similar to [Zuc96]. We restrict the summation to permutations $U = \pi$ with $\gamma(\pi) = 0$, i.e

$$per_{\gamma(U)=0}(M) = \sum_{\pi\in\mathcal{S}_n,\gamma(\pi)=0} \prod_{i=1}^{n} m_{i,\pi(i)}$$

(and analogously for $ham_{\gamma(U)=0}(M)$). Clearly, both $per_{\gamma(U)=0}(M)$ and

$ham_{\gamma(U)=0}(M)$ are in $CGen\mathbf{P}_{\mathbb{R}}$. Similarly, we modify TSP_{cost} to obtain $TSP_{cost}(\gamma(U) = 0)$: Given an edge weighted graph, compute the minimum cost of a hamiltonian tour under the additional constraint $\gamma(U) = 0$.

We obtain

Theorem 1. *$per_{\gamma(U)=0}(M)$ and $ham_{\gamma(U)=0}(M)$ are $CGen\mathbf{P}_{\mathbb{R}}$-complete.*

and

Theorem 2. *$TSP_{cost}(\gamma(U) = 0)$ is $CMin\mathbf{P}_{\mathbb{R}}$-complete.*

We also show

Theorem 3. *$ham(M)$ is $Gen\mathbf{P}_{\mathbb{R}}$-complete and TSP_{cost} is $Min\mathbf{P}_{\mathbb{R}}$-complete.*

However, it remains open whether $per(M)$ is $Gen\mathbf{P}_{\mathbb{R}}$-complete.

Contrary to $\sharp\mathbf{P}_{\mathbb{R}}$ the class $Gen\mathbf{P}_{\mathbb{R}}$ captures the permanent computation of a matrix M with arbitrary entries. However, there is a major other difference

between the two classes which implies that the former is not a subclass of the latter. In $\sharp\mathbf{P}_{\mathbb{R}}$ the counting is done in relation with non-combinatorial features, i.e. the corresponding functions count satisfying guesses within a $\mathbf{NP}_{\mathbb{R}}$ verification procedure. The latter guesses in general are vectors of reals. This is not possible within $Gen\mathbf{P}_{\mathbb{R}}$. In section 4 we therefore show how to enlarge the definition of generating functions once more to capture also counting processes as they are present in $\sharp\mathbf{P}_{\mathbb{R}}$. The class $Met\text{-}Gen\mathbf{P}_{\mathbb{R}}$ is obtained that way; it generalizes both $\sharp\mathbf{P}_{\mathbb{R}}$ and $Gen\mathbf{P}_{\mathbb{R}}$.

We consider the feasibility problem F_{zero} which stands for all real polynomials having a real zero and show

Theorem 4. *Let $\mathcal{D}$ be a $\mathbb{R}$-structure with two weight functions t_1, t_2 representing two polynomials of degrees 4 and k resp., in the same number of unknowns (cf. example 1 below).*

The function $\sum_{(\mathcal{D},t_1,z)\in F_{zero}} \sum_{\underline{x}\in A^k} t_2(\underline{x}) \prod_{a\in\underline{x}} z(a)$ *is complete in $Met\text{-}Gen\mathbf{P}_{\mathbb{R}}$ under reductions in $\mathbf{P}_{\mathbb{R}}$ (where the condition under the first summation refers to the real zeros of that polynomial whose coefficients are represented by t_1.)*

This paper opens an avenue to classify the complexity of combinatorial functions in the BSS-model which are in $EXPT_{\mathbb{R}}$ on the one hand side and a kind of non-combinatorial generalization on the other. We exemplify our approach with a wide class of generating functions of graph properties and cost optimization problems in Linear Programming. The novelty of our approach consists in the identification of such functions as complete in their respective setting. The results are not really surprising, but neither are they obvious.

In this extended abstract we will focus on a rigorous proof for theorem 4 only. The proofs of the other statements are similar in spirit, but of course some care has to be taken. They will be given in the full version of this paper.

2 Background on $\mathbb{R}$-Structures

In this subsection we recall the basic notion of an $\mathbb{R}$-structure. It is a special case of so called meta-finite structures introduced in [GG98]. $\mathbb{R}$-structures were first analyzed in [GM96].

We suppose the reader familiar with the main terminology of logic as well as with the concepts of vocabulary, first-order formula or sentence, interpretation and structure (see for example [EF95]).

Let $\mathbb{R}^\infty$ denote the set of finite sequences of real numbers, i.e. $\mathbb{R}^\infty = \bigoplus_{k\in\mathbb{N}} \mathbb{R}^k$.

Definition 1. Let L_s, L_f be finite vocabularies where L_s may contain relation and function symbols, and L_f contains function symbols only. A $\mathbb{R}$-*structure of signature* $\sigma = (L_s, L_f)$ is a pair $\mathcal{D} = (\mathcal{A}, \mathcal{F})$ consisting of

(i) a finite structure $\mathcal{A}$ of vocabulary L_s, called the *skeleton* of $\mathcal{D}$, whose universe A will also be said to be the *universe* of $\mathcal{D}$, and
(ii) a finite set $\mathcal{F}$ of functions $X : A^k \to \mathbb{R}$ interpreting the function symbols in L_f.

Definition 2. *Let $\mathcal{D}$ be a $\mathbb{R}$-structure with skeleton $\mathcal{A}$. We denote by $|A|$ and also by $|\mathcal{D}|$ resp. the cardinality of the universe A of $\mathcal{A}$. This number is called the size of the structure $\mathcal{D}$. A $\mathbb{R}$-structure $\mathcal{D} = (\mathcal{A}, \mathcal{F})$ is* ranked *if there is a unary function symbol $r \in L_f$ whose interpretation ρ in $\mathcal{F}$ bijects A with $\{0, 1, ..., |A| - 1\}$. The function ρ is called* ranking. *We will write $i < j$ for $i, j \in A$ iff $\rho(i) < \rho(j)$. A k-ranking on A is a bijection between A^k and $\{0, 1, ..., |A|^k - 1\}$. It can easily be defined if a ranking is available. We denote by ρ^k the interpretation of the k-ranking induced by ρ.*

Throughout this paper we suppose all $\mathbb{R}$-structures to be ranked. We therefore notationally suppress the symbol $\leq$ in the sets $\mathcal{F}$ considered.

Example 1 ([GM96]). Let us see how to describe a real polynomial of degree at most 4 as a $\mathbb{R}$-structure.

Consider the signature $(\emptyset, \{r, c\})$ where the arities of r and c are 1 and 4 respectively, and require that r is interpreted as a ranking.

Let $\mathcal{D} = (\mathcal{A}, \mathcal{F})$ be any $\mathbb{R}$-structure where $\mathcal{F}$ consists of interpretations $C : A^4 \to \mathbb{R}$ and $\rho : A \to \mathbb{R}$ of c and r. Let $n = |A| - 1$ so that ρ bijects A with $\{0, 1, ..., n\}$. Then $\mathcal{D}$ defines a homogeneous polynomial $\widehat{g} \in \mathbb{R}[X_0, ..., X_n]$ of degree four, namely

$$\widehat{g} = \sum_{(i,j,k,\ell) \in A^4} C(i, j, k, \ell) X_i X_j X_k X_\ell.$$

We obtain an arbitrary, that is, not necessarily homogeneous, polynomial $g \in \mathbb{R}[X_1, ..., X_n]$ of degree four by setting $X_0 = 1$ in $\widehat{g}$. We also say that $\mathcal{D}$ defines g. Notice that for every polynomial g of degree four in n variables there is a $\mathbb{R}$-structure $\mathcal{D}$ of size $n + 1$ such that $\mathcal{D}$ defines g.

Clearly, this example can easily be extended to $\mathbb{R}$-structures which represent systems of polynomials. In section 4 we are in particular interested in structures giving two polynomials in the same number of variables.

3 Generating Functions of Graph Properties

We will shortly define the concenpt of generating functions of graph properties. Full details can be found in [Bür99].

Consider an edge-weigthed graph $G = (V, E, t)$, that is a graph together with a weight function $t : E \to \mathbb{R}$. For a subset $\hat{E}$ of E we extend t to be $t(\hat{E}) := \prod_{v \in \hat{E}} t(v)$.

Generating functions are now defined based on graph properties $\mathcal{E}$.

Definition 3. *Given a graph property $\mathcal{E}$ the generating function $Gen_{\mathcal{E}}$ assigns to every edge-weighted graph $G = (V, E, t)$ the value*

$$Gen_{\mathcal{E}}(G) \ := \ \sum_{\hat{E} \subset E} t(\hat{E}) \ ,$$

where the sum is taken over all subsets $\hat{E}$ such that the graph $(V, \hat{E})$ has property $\mathcal{E}$.

As already indicated in the introduction this definition can be modified in (different) straightforward manners such as taking a minimum. We are in particular interested in looking at G as a metafinite structure (see below) and varying the way property $\mathcal{E}$ is depending on it.

Important examples of generating functions are the ones explained in the introduction (permanent, hamiltonian etc.)

The theory as well can be extended to R-structures where R is an arbitrary ring. In [Mak00] $R = \mathbb{R}[\bar{X}]$ the polynomial ring over $\mathbb{R}$ in the variables $X_1, X_2, ...$ Typical generating functions in this case are Tutte polynomials, Jones polynomials and Kauffman brackets.

4 Non-combinatorial Counting

In this section we will further generalize the previously defined concepts. So far, the counting operation related to our generating functions was of combinatorial kind. More precisely, for a given $\mathbb{R}$-structure $\mathcal{D}$ the summation is taken over all U such that $(\mathcal{D}, U) \in \mathcal{E}$. Here U is a relation over the finite universe A of $\mathcal{D}$. Thus, only finitely many valid assignments for U exist. For each of them a real number term then is evaluated.

In [Mee97] a real counting class $\sharp\mathbf{P}_{\mathbb{R}}$ is defined. It is given as all functions f such that the values $f(x)$ correspond to the number of accepting guesses for a $\mathbf{NP}_{\mathbb{R}}$ machine M with input $x \in \mathbb{R}^{\infty}$.

Definition 4. *The class $\sharp\mathbf{P}_{\mathbb{R}}$ is given by all functions $f : \mathbb{R}^{\infty} \to \{0,1\}^{\infty} \cup \{\infty\}$ such that there exists a BSS-machine M working in polynomial time and a polynomial q satisfying*

$$f(y) = |\{z \in \mathbb{R}^{q(size(y))} | M(y, z) \text{ is an accepting computation}\}| \ .$$

The major difference between functions in $\sharp\mathbf{P}_{\mathbb{R}}$ and the generating functions defined above is the dependence of the former on real number guesses. From the point of view of the BSS model this difference is due to the fact that the decision problem: " is there a U such that $(\mathcal{D}, U) \in \mathcal{E}$?" belongs to the class $\mathbf{DNP}_{\mathbb{R}}$. This class is a subclass of $\mathbf{NP}_{\mathbb{R}}$ and denotes those problems in $\mathbf{NP}_{\mathbb{R}}$ where the verification procedure makes use of digital guesses (i.e. zeros and ones) only. The general power of $\mathbf{NP}_{\mathbb{R}}$ allows to guess arbitrary reals. Problems in $\mathbf{DNP}_{\mathbb{R}}$ can be decided by simple enumeration of the finitely many valid guesses. Therefore, any

member of $Gen\mathbf{P}_{\mathbb{R}}$ or $CGen\mathbf{P}_{\mathbb{R}}$ can be computed in $EXPT_{\mathbb{R}}$. For functions in $\sharp\mathbf{P}_{\mathbb{R}}$ the latter result is more complicated and related to quantifyer elimination. However, these functions only compute natural numbers. We will now extend generating functions once again. The class of functions obtained will capture both $CGen\mathbf{P}_{\mathbb{R}}$ and $\sharp\mathbf{P}_{\mathbb{R}}$. To clarify the ideas let us start with the following example.

Example 2. i) Consider again the permanent function of a $(n \times n)$ matrix M. Its evaluation can also be described in the following manner. Let p be a polynomial in n unknowns such that $p(x_1, ..., x_n) = 0$ if and only if $(x_1, ..., x_n)$ gives a permutation π of $\{1, ..., n\}$ (i.e. $x_i \in \{1, ..., n\}$ and $x_i \neq x_j$ for $i \neq j$).

Then $per(M) = \sum_{x, p(x)=0} M(x)$. Here, $M(x)$ is a polynomial giving for a permutation π the value $\prod_{i=1}^{n} m_{i,\pi(i)}$. It can easily be defined as Lagrange polynomial. Computation of the permanent thus can be seen as evaluating a polynomial at all the zeros of another one and summing up the results.

ii) Let $f : \mathbb{R}^n \to \mathbb{R}$ be a polynomial bounded from below which should be minimized. Under the assumption that the usual first-order conditions of optimization theory are applicable one can look for the values of f at its critical points, i.e. on the zeros of its derivative $p := Df$. The problem gets the form of building the minimum of a polynomial on the zeros of another one. It can be generalized straightforwardly to constraint optimization problems.

The major difference between items i) and ii) above is that the zeros of p in the first part always consist of (small) integer components. In the second case the zeros may consist of real components, which even might not be computable. Moreover, the evaluation process in ii) has a much more general flavour and captures also the functions in $\sharp\mathbf{P}_{\mathbb{R}}$.

Note that evaluating polynomials on the zeros of other polynomials is also crucial in many quantifyer elimination procedures, where in particular the signs one polynomial takes in the zeros of another are of importance, see [R92], [CR88].

We define the class $Met\text{-}Gen\mathbf{P}_{\mathbb{R}}$ along the lines of the second example. Some care has to be taken. Consider a $\mathbb{R}$-structure $\mathcal{D}$ consisting of two weight functions t_1, t_2 together with a property $\mathcal{E} \in \mathbf{NP}_{\mathbb{R}}$. We want to build sums of the form $\sum_{(\mathcal{D},U,z)\in\mathcal{E}} T$ where U denotes a relation of fixed arity over the finite universe A of $\mathcal{D}$ and z represents a function from A to $\mathbb{R}$ (for simplicity assume z to be unary). Here, $\mathcal{E}$ depends on one of $\mathcal{D}$'s weight functions, whereas T depends on the other (in a way which has to be precised).

Since $\mathcal{E} \in \mathbf{P}_{\mathbb{R}}$ and z can be represented by a vector of $|A|$ many reals the decision problem: " is there a tuple (U, z) such that $(\mathcal{D}, U, z) \in \mathcal{E}$" is in $\mathbf{NP}_{\mathbb{R}}$. Moreover, there might be infinitely many valid assignments for (U, z) (i.e. for the z part). So the above summation can turn into an infinite series. In general, questions of convergence of such series are not decidable in the BSS model. Similarly, a function like $\min_{(\mathcal{D},z)\in\mathcal{E}} T(\mathcal{D}, z)$ is not correctly defined if the minimum doesn't ex-

ist. The question whether there are only finitely many satisfying assignments for z is decidable in the BSS framework; it corresponds to the question whether for a function $f \in \sharp\mathbf{P}_{\mathbb{R}}$ a value $f(x)$ is finite (and therefore exponentially bounded in the size of x, see [Mee97]).

We will define our generalized generating functions only for such $\mathbb{R}$-structures $\mathcal{D}$ for which $\sharp\{(U,z)|(\mathcal{D},U,z) \in \mathcal{E}\}$ is finite. Then sums as well as minimas are well defined. Examples which can be covered in this framework include polynomial optimization with finitely many Karush-Kuhn-Tucker points.

Another problem might be the lacking computability of all the valid guesses. Here, we want to restrict ourselves to such structures where computability is possible. The completeness result below holds true in general, but it can be restricted to those structures as well. If the assumption about computability is missing the evaluation problem gets a completely different touch, see the remarks at the end of this section.

Let $\mathcal{E}$ be a property in $\mathbf{P}_{\mathbb{R}}$. We consider $\mathbb{R}$-structures $\mathcal{D}$ having two weight functions t_1 and t_2 from $\mathcal{D}$'s finite universe to the reals. We assume the property $\mathcal{E}$ to depend on t_1 and the underlying finite structure of $\mathcal{D}$ only, i.e. $(\mathcal{D},U,z) \in \mathcal{E} \Leftrightarrow (\mathcal{D}^*,U,z) \in \mathcal{E}$ where $\mathcal{D}^*$ denotes the $\mathbb{R}$-structure obtained from $\mathcal{D}$ by removing the second weight function t_2.

As explained above we will only consider such structures $\mathcal{D}$ for which there are only finitely many valid guesses (U,z) such that $(\mathcal{D},U,z) \in \mathcal{E}$. These guesses moreover are assumed to be computable.

Definition 5. *a) The class Met-Gen$\mathbf{P}_{\mathbb{R}}$ is the set of all functions*

$$\sum_{(\mathcal{D},U,z)\in\mathcal{E}} \sum_{\bar{x}\in U} t_2(\bar{x}) \prod_{a\in\bar{x}} z(a)$$

where $\mathcal{D}$ and $\mathcal{E}$ are as assumed above. The condition "$a \in \bar{x}$" asks a to be a component of $\bar{x}$.

b) The class MinMet-Gen$\mathbf{P}_{\mathbb{R}}$ is obtained by taking

$$\min_{(\mathcal{D},U,z)\in\mathcal{E}} \sum_{\bar{x}\in U} t_2(\bar{x}) \prod_{a\in\bar{x}} z(a) \ .$$

Lemma 1. *a) CGen$\mathbf{P}_{\mathbb{R}} \subseteq$ Met-Gen$\mathbf{P}_{\mathbb{R}}$*

b) Any function in $\sharp\mathbf{P}_{\mathbb{R}}$ which only takes finite values belongs to Met-Gen$\mathbf{P}_{\mathbb{R}}$.

Both results are to be understood with respect to slight modifications of the inputted structures in such a way that they fit into the framework of input structures for functions in Met-Gen$\mathbf{P}_{\mathbb{R}}$.

Proof.

ad a) Consider a function

$$CGen_{\mathcal{E}}(\langle A, R^A\rangle, t_1) = \sum_{\langle A,R^A,U\rangle\in\mathcal{E}} \sum_{\bar{x}\in U} t_1(\bar{x})$$

with $\mathcal{E} \in \mathbf{P}_{\mathbb{R}}$.

We can artificially enlarge the property $\mathcal{E}$ by an additional component z for the guess in such a way that only the assignment 1 for all components of z will result in a valid guess (if the U-component is correct). The property thus obtained remains in $\mathbf{P}_{\mathbb{R}}$. Next, we enlarge a given $\mathbb{R}$-structure $\mathcal{D}$ by one further weight term t_2 which is equal to t_1. The corresponding evaluation on the enlarged structure gives the same value as the evaluation of $CGen_{\mathcal{E}}$ on $\mathcal{D}$.

ad b) Suppose $\mathcal{E} \in \mathbf{P}_{\mathbb{R}}$ and $f \in \sharp\mathbf{P}_{\mathbb{R}}$ such that $f(\mathcal{D}) = \#\{z|(\mathcal{D}, z) \in \mathcal{E}\} < \infty$. Without loss of generality we can suppose f to count satisfying guesses z of the form $z : A \to \mathbb{R}$ only, i.e. the finite relation U is captured as part of the real vector to be guessed (see [Mee97]).

We enlarge $\mathcal{D}$'s universe $A := \{1, ..., n\}$ to $\hat{A} := A \cup \{n + 1\}$. To $\mathcal{D}$ there corresponds the new structure $\hat{\mathcal{D}}$; it has universe $\hat{A}$ and the interpretations of the function and relation symbols correspond to those given by $\mathcal{D}$ on arguments from A^k and are zero (resp. not in the relation) if a component $n + 1$ is present. A new weight term $t_2 : \hat{A} \to \mathbb{R}$ is defined by $t_2(i) = \ \forall 1 \leq i \leq n$ and $t_2(n + 1) = 1$.

The property $\mathcal{E}$ is modified to $\hat{\mathcal{E}}$ by defining $(\hat{\mathcal{D}}, \hat{U}, \hat{z}) \in \hat{\mathcal{E}}$ iff $\hat{U} = \{n+1\}, \hat{z} = (z, 1)$ and $(\mathcal{D}, z) \in \mathcal{E}$. We thus obtain

$$\sum_{(\hat{\mathcal{D}},\hat{U},\hat{z})\in\hat{\mathcal{E}}} \sum_{x\in\hat{U}} t_2(x) \prod_{a\in x} \hat{z}(a) \ = \sum_{(\mathcal{D},z)\in\mathcal{E}} t_2(n+1)\cdot\hat{z}(n+1) \ = \sum_{(\mathcal{D},z)\in\mathcal{E}} 1 = f(\mathcal{D})\,.$$

□

In [Mee97] it is shown that every function in $\sharp\mathbf{P}_{\mathbb{R}}$ is computable in simply exponential time in the BSS model. The proof can be applied to show the existence of complete members in $Met\text{-}Gen\mathbf{P}_{\mathbb{R}}$.

Consider a family of $\mathbb{R}$-structures $\mathcal{D}$ representing two multivariate polynomials t_1 and t_2, the first of which is of degree at most 4 and the second of degree $k, k \in \mathbb{N}$, see example 1. Both polynomials depend on the same number of variables.

Theorem 4. *Let $\mathcal{E} := F_{zero}$ be the set of structures representing a polynomial having a real zero.*

The (non-combinatorial) generating function

$$NGen_{\mathcal{E}}(\mathcal{D}, t_1, t_2) \ = \sum_{(\mathcal{D},z)\in F_{zero}} \sum_{\bar{x}\in A^k} t_2(\bar{x}) \prod_{a\in\bar{x}} z(a)$$

is complete in Met-Gen$\mathbf{P}_{\mathbb{R}}$ w.r.t. reductions in $\mathbf{P}_{\mathbb{R}}$. The condition $(\mathcal{D}, z) \in F_{zero}$ is to be understood w.r.t. the weight function t_1 of $\mathcal{D}$.

That is, evaluating a polynomial on the zeros of another one and summing up the results is a complete function in Met-Gen$\mathbf{P}_{\mathbb{R}}$ (under the assumptions made in relation with the definition of Met-Gen$\mathbf{P}_{\mathbb{R}}$).

Proof. Consider a property $\tilde{\mathcal{E}} \in \mathbf{P}_{\mathbb{R}}$ together with an input structure $(\tilde{\mathcal{D}}, \tilde{t_1}, \tilde{t_2})$ for the function generated by $\tilde{\mathcal{E}}$. The finite universe of $\tilde{\mathcal{D}}$ is $\tilde{A} := \{1, ..., n\}$.

The function value on the input $\tilde{\mathcal{D}}$ is

$$\sum_{(\tilde{\mathcal{D}}, \tilde{U}, \tilde{z}) \in \tilde{\mathcal{E}}} \sum_{\bar{x} \in \tilde{U}} t_2(\bar{x}) \prod_{a \in \bar{x}} \tilde{z}(a) \ .$$

The problem: " is there a $(\tilde{U}, \tilde{z})$ such that $(\tilde{\mathcal{D}}, \tilde{U}, \tilde{z}) \in \tilde{\mathcal{E}}$ " belongs to $\mathbf{NP}_{\mathbb{R}}$ and thus can be reduced in polynomial time to the problem of deciding whether a polynomial T_1 of degree at most 4 has a real zero (for notational simplicity we denote the polynomial given via its coefficient function t_1 by T_1). This reduction is not parsimonious; nevertheless, the following holds true according to [Mee97]: for every valid assignment $(\tilde{U}, \tilde{z})$ such that $(\tilde{\mathcal{D}}, \tilde{U}, \tilde{z}) \in \tilde{\mathcal{E}}$ we can compute a polynomial T_1 in variables $y := (u_1, ..., u_{n^k}, z, w_1, ..., w_l, v_1, ..., v_m)$ such that for the choice $(u_1, ..., u_{n^k}) := \tilde{U}$ and $z := \tilde{z}$ there are exactly 2^m many zeros of T_1. More precisely:

- the components $\tilde{w_1}, ..., \tilde{w_l}$ are uniquely determined by $(\tilde{U}, \tilde{z})$ whereas for every component v_i there are exactly two possible choices such that any of them results in a zero of T_1 (if we fix $(\tilde{U}, \tilde{z}, \tilde{w})$ as first components).
- the identification of a relation $\tilde{U}$ with the n_k variables $u_1, ..., u_{n^k}$ is via the natural order of $\tilde{A}$ according to the natural ranking on $\{1, ..., n\}$; i.e. $u_i = 1$ iff the i-th element of $\tilde{A}^k$ is in $\tilde{U}$, and $u_i = 0$ otherwise.

We are going to define a $\mathbb{R}$-structure $\mathcal{D}$ representing two polynomials. The first of these polynomials will be T_1 as described above. Thus, the universe A of $\mathcal{D}$ is taken to be the disjoint union of the sets $A_1 := \{1, ..., n^k\}$ (for the variables u_i), $A_2 := \{1, ..., n\}, A_3 := \{1, ..., \ell\}$, and $A_4 := \{1, ..., m\}$ (for the components of the variables z, w, and v respectively).

Next, the second polynomial T_2 included into $\mathcal{D}$ (given by its coefficient function t_2) has to be defined. If $\tilde{U}$ is a relation of arity k on $\tilde{A}$ we define T_2 to be a polynomial of degree $k + 1$ on $|A|$ many real variables.

Even though formally T_2 depends on all variables u, z, w, v only particular monomials will have non-vanishing coefficients.

More precisely, a coefficient $t_2(x_0, x_1, ..., x_k)$ might be $\neq 0$ only if $x_0 \in A_1$ and $(x_1, ..., x_k) \in A_2^k$.

In that case we define

$$t_2(x_0, ..., x_k) := \frac{1}{2^m} \cdot \tilde{t}_2(x_1, ..., x_k) \ .$$

We claim that evaluating T_2 on the zeros of T_1 gives

$$\sum_{(\tilde{\mathcal{D}},\tilde{U},\tilde{z})\in\tilde{\mathcal{E}}} \sum_{\bar{x}\in\tilde{U}} \tilde{t}_2(\bar{x}) \prod_{a\in\bar{x}} z(a) \ .$$

Calculation shows

$$\sum_{(\mathcal{D},y)\in F^4} \sum_{\bar{x}\in A^{k+1}} t_2(\bar{x}) \prod_{a\in\bar{x}} y(a) \quad = \quad \sum_{y,T_1(y)=0} \sum_{\bar{x}\in A^{k+1}} t_2(\bar{x}) \prod_{a\in\bar{x}} y(a)$$

$$= \sum_{\substack{(u,z,w,v),\\ T_1(u,z,w,v)=0}} \sum_{(a_0,\bar{x})\in A_1\times A_2^k} t_2(a_0,\bar{x}) \prod_{a\in\bar{x}} u(a_0)\cdot z(a)$$

$$= \sum_{\substack{(u,z),(u,z) \text{ first}\\ \text{components of zero of } T_1}} 2^m \cdot \sum_{\bar{x}\in\tilde{U}} \tfrac{1}{2^m} \cdot \tilde{t}_2(\bar{x}) \prod_{a\in\bar{x}} z(a)$$

(here $\tilde{U}$ denotes the relation defined by those u_i which are 1)

$$= \sum_{(\tilde{\mathcal{D}},\tilde{U},z)\in\tilde{\mathcal{E}}} \sum_{\bar{x}\in\tilde{U}} \tilde{t}_2(\bar{x}) \prod_{a\in\bar{x}} z(a) \quad = \quad NGen_{\tilde{\mathcal{E}}}(\tilde{\mathcal{D}}) \ .$$

□

Remark 1. We could also allow counting according to functions $z : A^t \to \mathbb{R}$ for some arbitrary arity $t \geq 1$.

Similarly, in the above complete problem we could reduce the degree of T_2 to be at most 4 as well by applying the same reduction idea used for T_1. We omitted this in order not to get lost in details.

Complete problems for $MinMet\text{-}Gen\mathbf{P}_{\mathbb{R}}$ are obtained in the same way. As already mentioned, this class of functions includes the minimization of polynomial functions according to side constraints which are expressed via $\mathcal{E}$. For example, non-convex quadratic optimization problems with linear constraints can be described this way (cf. [Mee94]).

A completely new issue appears if we do not suppose the valid guesses of a $\mathbf{NP}_{\mathbb{R}}$-property to be computable for the given input (as is the case in many situations)!

Then, even if we assume the number of valid guesses to be finite the computation of the corresponding generating function cannot be done exactly. In that situation, we are led to approximating these guesses and then performing the subsequent evaluation also approximately. This is a completely different program than what we have done here; of course, the underlying questions of approximative computations are of high relevance. For results in relation with the BSS model confer [BCSS98].

In case we also want to get wid of the assumptions concerning the finiteness of the number of valid guesses one can think about a variation of the BSS model where also the evaluation of infinte series is possible. Steps into this direction can be found in [HSV97].

References

BCS97. P. Bürgisser, M. Clausen, and M.A. Shokrollahi. *Algebraic Complexity Theory*, volume 315 of *Grundlehren*. Springer Verlag, 1997.

BCSS98. L. Blum, F. Cucker, M. Shub, and S. Smale. *Complexity and Real Computation*. Springer Verlag, 1998.

Bür99. P. Bürgisser. *Completeness and reduction in algebraic complexity theory*, volume 7 of *Algorithms and Computation in Mathematics*, Springer Verlag, in press.

CR88. M. Coste and M.F. Roy, Thom's Lemma, the Coding of Real Algebraic Numbers and the Computation of the Topology of Semi-algebraic Sets. *J. Symbolic Computation*, 5, 121–129, 1988.

EF95. H.D. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer Verlag, 1995.

GG98. E. Grädel and Y. Gurevich. Metafinite model theory. *Information and Computation*, 140:26–81, 1998.

GM96. E. Grädel and K. Meer. Descriptive complexity theory over the real numbers. In: J. Renegar, M. Shub, and S. Smale (eds.): The Mathematics of Numerical Analysis. *Lectures in Applied Mathematics* 32, AMS, 381–404, 1996.

HSV97. G. Hotz, B. Schieffer and G. Vierke Analytic machines Technical Report TR95-025, ECCC, 1995.

Mak00. J.A. Makowsky. Colored Tutte Polynomials and Kauffman Brackets for Graphs of Bounded Tree Width. Extended Abstract , submitted MFCS'00; revised further, March 31, 2000, submitted to Combinatorics, Probability and Computation

Mee94. K. Meer. On the complexity of quadratic programming in real number models of computation. *Theoretical Computer Science*, 133:85–94, 1994.

Mee97. K. Meer. Counting problems over the Reals. To appear in *Theoretical Computer Science*. Extended abstract in: *Proc. of the 22nd international symposium on Mathematical Foundations of Computer Science MFCS*, LNCS 1295, Springer, 398–407, 1997.

R92. J. Renegar. On the computational Complexity and Geometry of the first-order Theory of the Reals , I - III. *Journal of Symbolic Computation*, vol. 13, 255–352, 1992.

SST95. S. Saluja, K. Subrahmanyam, and M. Thakur. Descriptive complexity of $\sharp p$ functions. *Journal of Computer and System Sciences*, 50:493–505, 1995.

Val79. L.G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.

Zuc96. D. Zuckerman. On unapproximable versions of NP-complete problems. *SIAM Journal on Computing*, 25(6):1305–1317, 1996.

Elimination of Negation in a Logical Framework

Alberto Momigliano

Department of Philosophy, Carnegie Mellon University, Pittsburgh, PA 15213, U.S.A.
mobile@cs.cmu.edu

Abstract. Logical frameworks with a logic programming interpretation such as hereditary Harrop formulae (HHF) [12] cannot express directly negative information, although negation is a useful specification tool. Since negation-as-failure does not fit well in a logical framework, especially one endowed with hypothetical and parametric judgments, we adapt the idea of *elimination* of negation introduced in [17] for Horn logic to a fragment of higher-order HHF. This entails finding a middle ground between the Closed World Assumption usually associated with negation and the Open World Assumption typical of logical frameworks; the main technical idea is to isolate a set of programs where static and dynamic clauses do not overlap.

1 Introduction

Deductive systems consist of axioms and rules defining derivable judgments; they can be used to specify logics and aspects of programming languages such as operational semantics or type systems. A *logical framework* is a meta-language for the specification, implementation and verification of deductive systems and possibly their meta-theory. A logical framework must provide tools which make encodings as simple and direct as possible. One well known example is higher-order abstract syntax, which moves renaming and substitution principles to the meta-language. Logical frameworks should be by design as weak as possible to simplify proofs of adequacy of encodings, effective checking of the validity of derivations and proof-search as well as unification. Many logical framework have been proposed in the literature (see [16] for an overview) and many extensions are also under consideration. However, we must carefully balance the benefits that any proposed extension can bring against the complications its meta-theory would incur.

This paper discusses the introduction of a logically justified notion of *negation* in logical frameworks with a logic programming interpretation such as hereditary Harrop formulae (HHF) [12] and its implementation in λProlog [15]. We intend this to form the basis for type-theoretic frameworks such as LF [9] and its implementation *Twelf* [19]. Those systems do not provide a *primitive* negation operator. Indeed, constructive logics usually implement negative information as $\neg A \equiv A \rightarrow \bot$, where $\bot$ denotes absurdity and the Duns Scoto Law is the elimination rule. Thus negative predicates have no special status; that would correspond to explicitly code negative information in a program, which is entirely consistent with the procedural interpretation of hypothetical judgments available in logical frameworks with a logic programming interpretation. However, this would not only significantly complicate goal-oriented

P. Clote and H. Schwichtenberg (Eds.): CSL 2000, LNCS 1862, pp. 411–426, 2000.

proof search, but providing negative definitions seems to be particularly error-prone, repetitive and not particularly interesting; more importantly, in a logical framework we have also to fulfill the *proof obligation* that the proposed negative definition does behave as the complement (of its positive counterpart). Automating the synthesis of negative information has not only an immediate practical relevance in the logic programming sense, but it may also have a rather dramatic effect on the possibility of implementing deductive systems that would prove to be too unwieldy to deal with otherwise. The synthesis of the negation of predicates such as *typable, well-formed, canonical form, subsort, value* etc.–as well as Prolog-like predicates such as equality, set membership and the like–will increase the amount of meta-theory that can be formalized.

Traditionally, negation-as-failure (*NF*) [5] has been the overwhelmingly used approach in logic programming (see [2] for a recent survey): that is, infer $\neg A$ if every proof of A fails finitely. The operational nature of this rule motivates the lack of a unique semantics and some of its related troublesome features: possible unsoundness, incompleteness and floundering. Furthermore, even if we manage to isolate a well-behaved logical fragment, such as acyclic normal programs, allowing *NF* in a logical framework would make adequacy theorems more difficult to prove, as both provability and unprovability must now be considered. The situation is even further complicated when we step to frameworks with hypothetical judgments; as recognized first by Gabbay [6], the unrestricted combination of *NF* and embedded implication is particularly problematic, since it leads to the failure of basic logic principles such as cut-elimination.

The approach to negation that we adopt is *transformational*, also known as *intensional negation*, initiated in [17] and developed in Pisa [3] for Horn logic with negation. Roughly, given a clause with occurrences of negated predicates, say $Q \leftarrow G, \neg P, G'$, where P is an already defined atom, the aim is to derive a *positive* predicate, say non_P which implements the complement of P, preserving operational equivalence; then, it is merely a question of replacement, yielding the negation-less clause $Q \leftarrow G, non\_P, G'$. This has the neat effect that negation and its problems are *eliminated*, i.e. we avoid any extension to the (meta) language. Technically, we can achieve this by transforming a Horn program into negation normal form and then by negating atoms via complementing terms, a problem first addressed in [10] for first-order terms. A final issue, which we do not tackle here, is dealing with *local* variables, which, during the transformation, become (extensionally) universally quantified [1].

Unfortunately, this approach does not scale immediately to logical frameworks such as HHF, for three main reasons:

1. The simply-typed λ-calculus is not closed under term complement.
2. Negation normal forms are incompatible with the operational semantics required by HHF.
3. There is an intrinsic tension between the *Closed World Assumption* (CWA), which is associated with negation, and the *Open World Assumption* (OWA) typical of languages with embedded implication.

The first problem has been solved in [14], by introducing a *strict* λ-calculus where term complement in the simply typed λ-calculus can be embedded and performed.

The second issue is orthogonal and requires an operational notion of normal form. The third one is rooted in the fundamental difference between Horn and HHF formulae: as well known, a Horn predicate definition can be seen as an inductive definition of the same predicate. The *minimality* condition of inductive definitions excludes anything else which is not allowed by the base and step case(s). This corresponds in Horn logic to the existence of the least model and to the consistency of the CWA and its finitary approximation, the *completion* of a program [5]: every atom which is not provable from a program is assumed to be false. Languages which provide embedded implication and universal quantification are instead *open-ended* and thus require the OWA; in fact, dynamic assumptions may, at run-time, extend the current signature and program in a totally unpredictable way. This makes it in general impossible to talk about the closure of such a program. In the literature the issue has been addressed in essentially three ways:

1. By enforcing a strict distinction between CWA and OWA predicates and applying *NF* only to the former [8], where the latter would require minimal negation.
2. By switching to a modal logic, which is able to take into account *arbitrary extensions* of the program as possible worlds (see the completion construction in [7] for N-Prolog and [4] for Hypothetical Datalog).
3. By embracing the idea of *partiality* in inductive definitions and using the rule of *definitional reflection* to incorporate a proof-theoretical notion of closure analogous to the completion [11].

None of those approaches are satisfactory for our purposes: most of the predicates we want to negate are open-ended; similarly, definitional reflection is not well-behaved (for example cut is not eliminable) for that very class of programs we are interested in. Moreover, we need to express the negation of a predicate in the same language where the predicate is formulated. Our solution is to restrict the set of programs we deem deniable in a novel way, so as to enforce a *Regular Word Assumption* (RWA): we define a class of programs whose dynamic assumptions extend the current database in a specific regular way. This constitutes a reasonable middle ground between the CWA which allows no dynamic assumption but is amenable to negation and the OWA, where assumptions are totally unpredictable. The RWA is also a promising tool in the study of the meta-logical frameworks [18]. Technically, this regularity under dynamic extension is calibrated so as to ensure that static and dynamic clauses never *overlap*. This property extends to the negative program; in a sense, we maintain a distinction between static and dynamic information, but at a much finer level, i.e. *inside* the definition of a predicate. The resulting fragment is very rich, as it captures the essence of the usage of hypothetical and parametric judgments in a logical framework; namely, that they are intrinsically combined to represent *scoping* constructs in the object language. This is why we contend that this class of programs is adequate for the practice of logical frameworks.

It is clear that elimination of negation makes sense only when negation is *stratified*, i.e. the negative predicates ultimately refers (in the call graph) to a positive one. While there may be a place in logic programming for non-stratified negation, this does not seem to be the case for a logical framework. Another difference from traditional logic programming is that negation applies only to *terminating* programs;

thus it refers not to finite failure but to unprovability tout court, as we refrain from negating programs whose negation is not recursively axiomatizable. We will thus identify negation with a *complement* operation.

The rest of the paper is organized as follow: in Sect. 2 we give an informal view of the complement algorithm by means of examples, while Sect. 3 introduces the language. Section 4 describe term and clause complementation. We conclude in Sect. 5 with some remarks on future work. We refer to [13] for more details and proofs omitted here for reasons of space.

2 A Motivating Example

Consider the expressions of the untyped λ-calculus:

$$e ::= x \mid \Lambda x \,.\, e \mid e_1\, e_2$$

We encode these expressions as terms in (labeled) HHF via the usual techniques of higher-order abstract syntax as canonical forms over the following signature:

$$\Sigma = exp : \text{type}, lam : (exp \to exp) \to exp, app : exp \to (exp \to exp)$$

The representation function is given by:

$$\ulcorner x \urcorner = x \qquad \ulcorner \Lambda x \,.\, e \urcorner = lam\ (\lambda x{:}exp.\ulcorner e \urcorner) \qquad \ulcorner e_1\ e_2 \urcorner = app\ \ulcorner e_1 \urcorner\ \ulcorner e_2 \urcorner$$

A term is *linear* if every functional subterm uses each argument exactly once: in particular, we check for linearity of a function making sure that the latter is linear in its first argument and then recurring on the rest of the expression.

$$\begin{aligned} linapp &: linear(app\ E_1\ E_2) \leftarrow linear(E_1) \wedge linear(E_2). \\ linlam &: linear(lam(\lambda x \,.\, E\ x)) \\ &\qquad \leftarrow linx(\lambda x \,.\, E\ x) \wedge (\forall y{:}exp.\, linear(y) \to linear(E\ y)). \end{aligned}$$

$$\begin{aligned} linxx &: linx(\lambda x \,.\, x). \\ linxap1 &: linx(\lambda x \,.\, app\ (E_1\ x)\ E_2) \leftarrow linx(\lambda x \,.\, E_1\ x). \\ linxap2 &: linx(\lambda x \,.\, app\ E_1\ (E_2\ x)) \leftarrow linx(\lambda x \,.\, E_2\ x). \\ linxlm &: linx(\lambda x \,.\, lam(\lambda y \,.\, E\ x\ y)) \leftarrow (\forall y{:}exp.\, linx(\lambda x \,.\, E\ x\ y)). \end{aligned}$$

This is clearly a decision procedure, which can be complemented; an expression is *not* linear if there is some function which either does not use its argument or uses it more than once. First, the complement of `linapp` does not pose any problem, as it is a Horn clause: an application is not linear if either the first element or the second is not linear. Next, a lambda expression is not linear in two cases: one, if it is not linear in its first argument:

$$\neg linlam1 : \neg linear(lam\ \lambda x \,.\, E\ x) \leftarrow \neg linx(\lambda x \,.\, E\ x).$$

Secondly, if its body is not linear. Now, this poses a new problem, as we have to negate a hypothetical and parametric goal. Let us reason by example and suppose we are given, in the empty context, a goal $linear(lam(\lambda x\,.\,lam(\lambda y\,.\,x)))$, which is unprovable, since the second lambda term is not linear in y; the proof tree yields the failure leaf $linx(\lambda x\,.\,z)$, for a new parameter z, in the context $z{:}exp; linear(z)$. Our guiding intuition is that we want to mimic a failure derivation so as to provide a successful derivation from the negative definition, i.e. a proof of $\neg linx(\lambda x\,.\,z)$ from $z{:}exp; linear(z)$; this shows one prominent feature of complementation of an HHF formula: negation 'skips' over $\forall$ and $\rightarrow$, since it needs to mirror failure *from* assumptions. Now, let us examine clause `linxlm` and reconsider the above failure leaf; in a first attempt, according to the idea above, the complement would be:

$$\overset{?}{\neg}\, linxlm : \neg linx(\lambda x\,.\,lam(\lambda y\,.\,E\ x\ y)) \leftarrow (\forall y{:}exp.\, \neg linx(\lambda x\,.\,E\ x\ y)).$$

However, there is no way to obtain a proof of $\neg linx(\lambda x\,.\,z)$ from the current context. Indeed, the `linxlm` clause does not carry enough information so that its complement can mimic the failure proof. In a sense, the clause is not *assumption-complete*: once it has introduced a new parameter, the clause only specifies how to use it in a positive context. It is up to us to synthesize its dynamic negative definition, in this case simply $\forall y{:}exp.\, \neg linx(\lambda x\,.\,y)$. More in general, it is a characteristic of HHF that the negation of a clause is not strong enough to determine the behavior of a program under complementation. We will have to insert (via a source-to-source transformation) additional structure in a predicate definition, in order to completely determine the provability or failure of goals which mention *parameters*. By observing the structure of all possible assumption that a predicate definition can make, we will *augment* those assumptions with their negative definition. In particular, we first augment the clause `linxlm`:

$$\begin{aligned} aug_{\mathrm{D}}(linxlm) : linx(\lambda x\,.\,lam(\lambda y\,.\,E\ x\ y)) \\ \leftarrow (\forall y{:}exp.\, \neg linx(\lambda x\,.\,y) \rightarrow linx(\lambda x\,.\,E\ x\ y)). \end{aligned}$$

so that, by complementation, we obtain:

$$\begin{aligned} \neg aug_{\mathrm{D}}(linxlm) : \neg linx(\lambda x\,.\,lam(\lambda y\,.\,E\ x\ y)) \\ \leftarrow (\forall y{:}exp.\, \neg linx(\lambda x\,.\,y) \rightarrow \neg linx(\lambda x\,.\,E\ x\ y)). \end{aligned}$$

Unfortunately, the procedure we have outlined is not possible in general. Consider a clause encoding the introduction rule for implication in natural deduction, which can be used to check whether an implicational formula trivially holds:

$$\begin{aligned} \Sigma = form : type, imp : form \rightarrow (form \rightarrow form), a : form, b : form, c : form \\ impi\ :\ nd(A\ imp\ B) \leftarrow (nd(A) \rightarrow nd(B)). \end{aligned}$$

Following our earlier remark its complement would be:

$$\begin{aligned} &\neg impi1 : \neg nd(a). \\ &\neg impi2 : \neg nd(b). \\ &\neg impi3 : \neg nd(c). \\ &\overset{?}{\neg}\, impi : \neg nd(A\ imp\ B) \leftarrow (nd(A) \rightarrow \neg nd(B)). \end{aligned}$$

This specification is clearly incorrect since both $nd(a\ imp\ a)$ **and** $\neg nd(a\ imp\ a)$ are derivable from the empty context. We can isolate one major problem: in clause `impi` the assumption $nd(A)$ which is dynamically added to the (static) definition of the `nd` predicate *overlaps* with the head of the clause. A symmetrical problem can occur when dynamic and static clause do differ but their complements do not. We have thus isolated two main issues:

1. Exhaustivity: we need to enrich clauses so that every (ground) goal or its negation is provable.
2. Exclusivity: we need to isolate a significant fragment where it is not the case that both a goal and its negation are provable.

We will achieve exhaustivity (Theorem 2) by *augmenting* the program with the complement of assumptions; moreover, we will achieve exclusivity (Theorem 1) with the restriction to *complementable* programs. To anticipate the idea, a clause is complementable if every assumption contains some eigenvariable at execution time.

3 Provability and Unprovability

We will use the following somewhat unusual language:

$$\begin{array}{rrl}
\textit{Simple Types} & A & ::= a \mid A_1 \to A_2 \\
\textit{Terms} & M & ::= c \mid x \mid \lambda x{:}A.\ M \mid M_1\ M_2 \\
\textit{Atoms} & Q & ::= q\ \overline{M_n} \mid \neg q\ \overline{M_n} \\
\textit{Clauses} & D & ::= \top \mid \bot \mid Q \leftarrow G \mid D_1 \wedge D_2 \mid D_1 \vee D_2 \mid \forall x{:}A.\, D \\
\textit{Goals} & G & ::= Q \mid \top \mid \bot \mid \overline{M_n} \doteq \overline{N_n} \mid \overline{M_n} \neq \overline{N_n} \mid \\
 & & \quad G_1 \wedge G_2 \mid G_1 \vee G_2 \mid D \to G \mid \forall x{:}A.\, G \\
\textit{Signatures} & \Sigma & ::= \cdot \mid \Sigma, a{:}\mathrm{type} \mid \Sigma, c{:}A \\
\textit{Parameter Contexts} & \Gamma & ::= \cdot \mid \Gamma, x{:}A \\
\textit{Assumptions} & \mathcal{D} & ::= \top \mid \mathcal{D} \wedge D
\end{array}$$

There is a distinguished type **o** for propositions which can occur only as the target of some A. We remark that '$\neg$' is *not* a connective, but a name constructor for atomic formulae; 'facts' are represented, for convenience, by $Q \leftarrow \top$, although in examples we will omit to mention $\top$. We assume that existential variables occur only once in the head of program clauses (i.e. clauses are *left-linear*); this can always be achieved by introducing disequations in the body. In this paper we restrict ourselves to programs such that all assumptions are Horn and which can be proven to be terminating under some well-founded ordering. We introduce the uniform proofs system [12] for (immediate) provability and denial in Fig. 1. For terminating programs, we can prove that the failure to achieve a proof of G translates into (a derivation of) the denial of G. Note also that due to the presence of disjunction as a clause constructor, uniform proofs are *not* complete for our language. We will remedy this situation in Sect. 4.

$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} G$ Program $\mathcal{P}$ and assumption $\mathcal{D}$ uniformly entail G.
$\Gamma; \mathcal{D} \nvdash_{\mathcal{P}} G$ Program $\mathcal{P}$ and assumption $\mathcal{D}$ uniformly deny G.
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} D \gg Q$ Clause D from $\mathcal{P}$ and $\mathcal{D}$ immediately entails atom Q.
$\Gamma; \mathcal{D} \nvdash_{\mathcal{P}} D \gg Q$ Clause D from $\mathcal{P}$ and $\mathcal{D}$ immediately denies atom Q.

$$\frac{}{\Gamma;\mathcal{D}\vdash_{\mathcal{P}}\top}\vdash\top \qquad \frac{}{\Gamma;\mathcal{D}\nvdash_{\mathcal{P}}\bot}\nvdash_{\bot}$$

$$\frac{\overline{M_n}=\overline{N_n}}{\Gamma;\mathcal{D}\vdash_{\mathcal{P}}\overline{M_n}\doteq\overline{N_n}}\vdash\doteq \qquad \frac{\overline{M_n}\neq\overline{N_n}}{\Gamma;\mathcal{D}\nvdash_{\mathcal{P}}\overline{M_n}\doteq\overline{N_n}}\nvdash\doteq$$

$$\frac{\overline{M_n}\neq\overline{N_n}}{\Gamma;\mathcal{D}\vdash_{\mathcal{P}}\overline{M_n}\mathrel{\dot{\neq}}\overline{N_n}}\vdash\dot{\neq} \qquad \frac{\overline{M_n}=\overline{N_n}}{\Gamma;\mathcal{D}\nvdash_{\mathcal{P}}\overline{M_n}\mathrel{\dot{\neq}}\overline{N_n}}\nvdash\dot{\neq}$$

$$\frac{\Gamma;\mathcal{D}\vdash_{\mathcal{P}}G_1 \qquad \Gamma;\mathcal{D}\vdash_{\mathcal{P}}G_2}{\Gamma;\mathcal{D}\vdash_{\mathcal{P}}G_1\wedge G_2}\vdash\wedge \qquad \frac{\Gamma;\mathcal{D}\nvdash_{\mathcal{P}}G_1 \qquad \Gamma;\mathcal{D}\nvdash_{\mathcal{P}}G_2}{\Gamma;\mathcal{D}\nvdash_{\mathcal{P}}G_1\vee G_2}\nvdash\vee$$

$$\frac{\Gamma;\mathcal{D}\vdash_{\mathcal{P}}G_i}{\Gamma;\mathcal{D}\vdash_{\mathcal{P}}G_1\vee G_2}\vdash\vee_i \qquad \frac{\Gamma;\mathcal{D}\nvdash_{\mathcal{P}}G_i}{\Gamma;\mathcal{D}\nvdash_{\mathcal{P}}G_1\wedge G_2}\nvdash\wedge_i$$

$$\frac{\Gamma\vdash t:A \qquad \Gamma;\mathcal{D}\vdash_{\mathcal{P}}[t/x]G}{\Gamma;\mathcal{D}\vdash_{\mathcal{P}}\exists x{:}A.\,G}\vdash\exists \qquad \frac{\text{for all } n\ \Gamma\vdash n:A \qquad \Gamma;\mathcal{D}\nvdash_{\mathcal{P}}[n/x]G}{\Gamma;\mathcal{D}\nvdash_{\mathcal{P}}\exists x{:}A.\,G}\nvdash\exists$$

$$\frac{\Gamma;(\mathcal{D}\wedge D)\vdash_{\mathcal{P}}G}{\Gamma;\mathcal{D}\vdash_{\mathcal{P}}D\to G}\vdash\to \qquad \frac{\Gamma;(\mathcal{D}\wedge D)\nvdash_{\mathcal{P}}G}{\Gamma;\mathcal{D}\nvdash_{\mathcal{P}}D\to G}\nvdash\to$$

$$\frac{(\Gamma,y{:}A);\mathcal{D}\vdash_{\mathcal{P}}[y/x]G}{\Gamma;\mathcal{D}\vdash_{\mathcal{P}}\forall x{:}A.\,G}\vdash\forall^{y} \qquad \frac{(\Gamma,y{:}A);\mathcal{D}\nvdash_{\mathcal{P}}[y/x]G}{\Gamma;\mathcal{D}\nvdash_{\mathcal{P}}\forall x{:}A.\,G}\nvdash\forall^{y}$$

$$\frac{\Gamma;\mathcal{D}\vdash_{\mathcal{P}}(\mathcal{P}\wedge\mathcal{D})\gg Q}{\Gamma;\mathcal{D}\vdash_{\mathcal{P}}Q}\vdash\mathrm{At} \qquad \frac{\Gamma;\mathcal{D}\nvdash_{\mathcal{P}}(\mathcal{P}\wedge\mathcal{D})\gg Q}{\Gamma;\mathcal{D}\nvdash_{\mathcal{P}}Q}\nvdash\mathrm{At}$$

$$\frac{}{\Gamma;\mathcal{D}\vdash_{\mathcal{P}}\bot\gg Q}\gg\bot \qquad \frac{}{\Gamma;\mathcal{D}\nvdash_{\mathcal{P}}\top\gg Q}\not\gg\top$$

$$\frac{\Gamma\vdash t:A \qquad \Gamma;\mathcal{D}\vdash_{\mathcal{P}}[t/x]D\gg Q}{\Gamma;\mathcal{D}\vdash_{\mathcal{P}}\forall x{:}A.\,D\gg Q}\gg\forall \qquad \frac{\text{for all } n\ \Gamma\vdash n:A \qquad \Gamma;\mathcal{D}\nvdash_{\mathcal{P}}[n/x]D\gg Q}{\Gamma;\mathcal{D}\nvdash_{\mathcal{P}}\forall x{:}A.\,D\gg Q}\not\gg\forall$$

$$\frac{\Gamma;\mathcal{D}\vdash_{\mathcal{P}}D_i\gg Q}{\Gamma;\mathcal{D}\vdash_{\mathcal{P}}D_1\wedge D_2\gg Q}\gg\wedge_i \qquad \frac{\Gamma;\mathcal{D}\nvdash_{\mathcal{P}}D_i\gg Q}{\Gamma;\mathcal{D}\nvdash_{\mathcal{P}}D_1\vee D_2\gg Q}\not\gg\vee_i$$

$$\frac{\Gamma;\mathcal{D}\vdash_{\mathcal{P}}D_1\gg Q \qquad \Gamma;\mathcal{D}\vdash_{\mathcal{P}}D_2\gg Q}{\Gamma;\mathcal{D}\vdash_{\mathcal{P}}D_1\vee D_2\gg Q}\gg\vee \qquad \frac{\Gamma;\mathcal{D}\nvdash_{\mathcal{P}}D_1\gg Q \qquad \Gamma;\mathcal{D}\nvdash_{\mathcal{P}}D_2\gg Q}{\Gamma;\mathcal{D}\nvdash_{\mathcal{P}}D_1\wedge D_2\gg Q}\not\gg\wedge$$

$$\frac{\overline{N_n}=\overline{M_n} \qquad \Gamma;\mathcal{D}\vdash_{\mathcal{P}}G}{\Gamma;\mathcal{D}\vdash_{\mathcal{P}}(q\ \overline{N_n}\leftarrow G)\gg q\ \overline{M_n}}\gg\to$$

$$\frac{\overline{N_n}\neq\overline{M_n}}{\Gamma;\mathcal{D}\nvdash_{\mathcal{P}}(q\ \overline{N_n}\leftarrow G)\gg q\ \overline{M_n}}\not\gg\to_1 \qquad \frac{\overline{N_n}=\overline{M_n} \qquad \Gamma;\mathcal{D}\nvdash_{\mathcal{P}}G}{\Gamma;\mathcal{D}\nvdash_{\mathcal{P}}(q\ \overline{N_n}\leftarrow G)\gg q\ \overline{M_n}}\not\gg\to_2$$

Fig. 1. (Immediate) Provability and Denial

Some brief comments are in order: the (in)equalities rules simply mirror the object logic symbols $\doteq, \neq$ as meta-level (in)equalities. $\not\gg\forall$ and $\not\vdash \exists$ are infinitary rules, given the meta-linguistic extensional universal quantification on all terms. Rules $\vdash \forall$, $\not\vdash \forall$ are instead parametric in y, where the $()^y$ superscript reminds us of the eigenvariable condition. The denial rules for implication and universal quantification reflect the operational semantics of unprovability that we have discussed earlier.

We start by putting every program in a *normalized* format w.r.t. assumptions, so that every goal in the scope of an universal quantifier is guaranteed to depend on some assumption, possibly the trivial clause $\top$. This has also the effect of 'localizing' the trivial assumption to its atom, a property will be central while complementing assumptions; for example we re-write `linxlm` as follows:

$$\begin{aligned} linxlm : linx\ (\lambda x\,.\,lam(\lambda y\,.\,E\ x\ y)) \\ \leftarrow (\forall x{:}exp.\,\top_{linx} \rightarrow linx\ (\lambda x\,.\,E\ x\ z)). \end{aligned}$$

For the sake of this paper, we also need to modify the source program so that every term in a clause head is *fully applied*, i.e. it is a lambda term where every variable mentioned in the binder occurs in the matrix; this makes term complementation (Sect. 4) much simpler. For example clause `linxap1` is rewritten as:

$$linxap1 : linx(\lambda x\,.\,app\ (E_1\ x)\ (E_2\ x)) \leftarrow linx(\lambda x\,.\,E_1\ x) \wedge vac(\lambda x\,.\,E_2\ x).$$

where $vac(\lambda x\,.\,E_2\ x)$ enforces that x does not occur in $E_2\ x$. Its definition is type-directed, but we have shown in [14] how to internalize these occurrence constraints in a strict type theory, so that this further transformation is not needed.

We now discuss context schemata. As we have argued in Sect. 2, we cannot obtain closure under clause complementation for the full logic of HHF, but we have to restrict ourselves to a smaller (but significant) fragment. This in turn entails that we have to make sure that during execution, whenever an assumption is made, it remains in the fragment we have isolated. Technically, we proceed as follows:

- We extract from the static definition of a predicate the general 'template' of a legal assumption.
- We require dynamic assumptions to conform to this template.

We thus introduce the notion of *schema satisfaction*, which uses the following data structure: a *context schema* abstracts over all possible instantiations of a context during execution. To account for that, we introduce a quantifier-like operator, say SOME $\Phi\,.\mathcal{D}$, which takes a clause and existentially bounds its free variables, if any, i.e. $\Phi = FV(\mathcal{D})$. The double bar '$\|$', not to be confused with the BNF '$|$' that we informally use in the meta-language, denotes schema alternatives, while '$\circ$' stands for the empty context schema.

$$Contexts\ Schemata\ \mathcal{S} ::= \circ \mid \mathcal{S}\|(\Gamma; \text{SOME}\ \Phi\,.\mathcal{D})$$

The `linear` predicate yields this (degenerate) example of context schema:

$$\mathcal{S}_{linear} = \circ \mid \mathcal{S}_{linear}\|x{:}exp; linear(x) \mid \mathcal{S}_{linear}\|x{:}exp; \top_{linx}$$

We extract a context schema by collecting all negative occurrences in a goal; this is achieved by simulating execution until an atomic goal is reached and the current list of parameters and assumptions is returned, with their correct existential binding. Different clauses may contribute different schema alternatives for a given predicate definition. A *run-time* context consists of a set of *blocks*, each of which is an instance of the context schema, for example:

$$y_1{:}exp, y_2{:}exp, x_1{:}exp; \top_{linx} \wedge \top_{linx} \wedge linear(x_1)$$

We will need to disambiguate blocks in run-time contexts; overlapping may indeed happen when the alternatives in a context schema are not disjoint. Intuitively, a block is complete when an atomic conclusion is reached during the deduction. Any bracketing convention will do:

$$\lceil y_1{:}exp \rceil, \lceil y_2{:}exp \rceil, \lceil x_1{:}exp \rceil; \lceil \top_{linx} \rceil \wedge \lceil \top_{linx} \rceil \wedge \lceil linear(x_1) \rceil$$

We then define when a formula satisfies a schema. We start by saying that a completed block *belongs* to a schema when the block is an alphabetic variant of some instantiation of one of the alternatives of the schema. Then, the empty run-time context is an *instance* of every schema. Secondly, if Γ' and $\mathcal{D}'$ are completed blocks which belong to $\mathcal{S}$, and $\Gamma; \mathcal{D}$ in an instance of $\mathcal{S}$, then $(\Gamma, \lceil \Gamma' \rceil); (\mathcal{D} \wedge \lceil \mathcal{D}' \rceil)$ is an *instance* of $\mathcal{S}$, provided that $\mathcal{D}'$ is a valid clause. The latter holds when each of its subgoals satisfies the schema. This is achieved by mimicking the construction on the run-time schema until in the base case we check whether the resulting context is an instance of the given schema.

We can prove that if a context schema is extracted from a program, then any instance of the latter satisfies the former. Moreover, execution preserves contexts, i.e. every subgoal which arises in any given successful or failed (immediate and non-immediate) sub-derivation satisfies the context schema. See [13] for the formal development.

4 Clause Complementation

We restrict ourselves to programs with:

- Goals where every assumption is parametric, i.e. it is in the scope of a positive occurrence of a universal quantifier and the corresponding parameter occurs in head position in the assumption.
- Clauses $Q \leftarrow G$ such that the head of every term in Q is rigid.

Note that the rigidity restriction applies only to non-Horn predicate definitions and can be significantly relaxed; see [13] for a detailed account.

The first ingredient is *higher-order pattern* complement, $\mathrm{Not}(M)$, investigated in the general case in [14]; we give here the rules for complementing fully applied patterns:

$$\frac{}{\Gamma \vdash \mathrm{Not}(E\ \overline{x_n}) \Rightarrow \emptyset}\ \mathrm{Not\_}Flx$$

$$\frac{\Gamma, x{:}A \vdash \mathrm{Not}(M) \Rightarrow N : B}{\Gamma \vdash \mathrm{Not}(\lambda x{:}A.\, M) \Rightarrow \lambda x{:}A.\, N : A \to B}\ \mathrm{Not\_}Lam$$

$$\frac{g \in \Sigma \cup \Gamma, g : A_1 \to \ldots \to A_m \to a, m \geq 0, h \not\equiv g}{\Gamma \vdash \mathrm{Not}(h\ \overline{M_n}) \Rightarrow g\ (Z_1\Gamma) \ldots (Z_m\Gamma) : a}\ \mathrm{Not\_}App^1$$

$$\frac{\exists i : 1 \leq i \leq n \qquad \Gamma \vdash \mathrm{Not}(M_i) \Rightarrow N :}{\Gamma \vdash \mathrm{Not}(h\ \overline{M_n}) \Rightarrow h\ (Z_1\Gamma) \ldots (Z_{i-1}\Gamma)\ N\ (Z_{i+1}\Gamma) \ldots (Z_n\Gamma) : a}\ \mathrm{Not\_}App^2$$

where the Z's are fresh variables which may depend on the domain of Γ, $h \in \Sigma \cup \Gamma$, and $\Gamma \vdash h : A_1 \to \ldots \to A_n \to a$. $\Gamma \vdash \mathrm{Not}(M) = \mathcal{N} : A$ iff $\mathcal{N} = \{N \mid \Gamma \vdash \mathrm{Not}(M) \Rightarrow N : A\}$. For example:

$$\cdot \vdash \mathrm{Not}(\lambda x\,.\,x) = \{\lambda x\,.\,lam(\lambda y\,.\,E\ x\ y), \lambda x\,.\,app\ (E_1\ x)\ (E_2\ x)\}$$

If we write $\Gamma \vdash M \in \|N\| : A$ when M is a ground instance of a pattern N at type A, we can show that Not behaves as the complement on sets of ground terms, i.e.

1. (Exclusivity) Not ($\Gamma \vdash M \in \|N\| : A$ and $\Gamma \vdash M \in \|\mathrm{Not}(N)\| : A$).
2. (Exhaustivity) Either $\Gamma \vdash M \in \|N\| : A$ or $\Gamma \vdash M \in \|\mathrm{Not}(N)\| : A$.

Complementing goals is immediate: we just put the latter in negation normal form, respecting the operational semantics of failure.

$$\frac{}{\mathrm{Not_G}(\top) = \bot}\ \mathrm{Not_G}\top \qquad \frac{}{\mathrm{Not_G}(\bot) = \top}\ \mathrm{Not_G}\ \bot \qquad \frac{}{\mathrm{Not_G}(Q) = \neg Q}\ \mathrm{Not_G}At$$

$$\frac{}{\mathrm{Not_G}(\overline{M_n} \doteq \overline{N_n}) = (\overline{M_n} \not\doteq \overline{N_n})}\ \mathrm{Not}\doteq \qquad \frac{}{\mathrm{Not_G}(\overline{M_n} \not\doteq \overline{N_n}) = (\overline{M_n} \doteq \overline{N_n})}\ \mathrm{Not}\not\doteq$$

$$\frac{\mathrm{Not_G}(G) = G'}{\mathrm{Not_G}(\forall x{:}A.\, G) = \forall x{:}A.\, G'}\ \mathrm{Not_G}\forall \qquad \frac{\mathrm{Not_G}(G) = G'}{\mathrm{Not_G}(D \to G) = D \to G'}\ \mathrm{Not_G} \to$$

$$\frac{\mathrm{Not_G}(G_1) = G_1' \qquad \mathrm{Not_G}(G_2) = G_2'}{\mathrm{Not_G}(G_1 \wedge G_2) = G_1' \vee G_2'}\ \mathrm{Not}\wedge$$

$$\frac{\mathrm{Not_G}(G_1) = G_1' \qquad \mathrm{Not_G}(G_2) = G_2'}{\mathrm{Not_G}(G_1 \vee G_2) = G_1' \wedge G_2'}\ \mathrm{Not}\vee$$

Clause complementation is instead more delicate: given a rule $q\ \overline{M_n} \leftarrow G$, its complement must contain a 'factual' part motivating failure due to clash with the head; the remainder $\mathrm{Not_G}(G)$ expresses failure in the body, if any. Clause complementation must discriminate whether (the head of) a rule belongs to the static or dynamic definition of a predicate. In the first case all the relevant information is already present in the head of the clause and we can use the term complementation algorithm. This is accomplished by the rule $\mathrm{Not_D} \to$, where a set of negative facts is built via term complementation $\mathrm{Not}(\overline{M_n})$, namely $\bigwedge_{\overline{N_n} \in \mathrm{Not}(\overline{M_n})} \forall(\neg q\ \overline{N_n} \leftarrow \top)$,

whose fresh free variables are universally closed; moreover the negative counterpart of the source clause is obtained via complementation of the body. The original quantification is retained thanks to rule $\mathrm{Not_D}\forall$.

$$\frac{}{\mathrm{Not_D}(\top) = \bot}\,\mathrm{Not_D}\top \qquad \frac{}{\mathrm{Not_D}(\bot) = \top}\,\mathrm{Not_D}\,\bot$$

$$\frac{\mathrm{Not_G}(G) = G'}{\mathrm{Not_D}(q\ \overline{M_n} \leftarrow G) = (\bigwedge_{\overline{N_n} \in \mathrm{Not}(\overline{M_n})} \forall(\neg(q\ \overline{N_n}) \leftarrow \top)) \wedge (\neg q\ \overline{M_n} \leftarrow G')}\,\mathrm{Not_D} \leftarrow$$

$$\frac{\mathrm{Not_D}(D) = D'}{\mathrm{Not_D}(\forall x\!:\!A.\ D) = \forall x\!:\!A.\ D'}\,\mathrm{Not_D}\forall$$

$$\frac{\mathrm{Not_D}(D_1) = D_1' \qquad \mathrm{Not_D}(D_2) = D_2'}{\mathrm{Not_D}(D_1 \wedge D_2) = D_1' \vee D_2'}\,\mathrm{Not}\wedge$$

$$\frac{\mathrm{Not_D}(D_1) = D_1' \qquad \mathrm{Not_D}(D_2) = D_2'}{\mathrm{Not_D}(D_1 \vee D_2) = D_1' \wedge D_2'}\,\mathrm{Not}\vee$$

Otherwise, we can think of the complement of an atomic assumption $(q\ M_1 \ldots x\ \ldots M_n)$, which is by definition parametric in some x, as static clause complementation w.r.t. x, i.e. $\mathrm{Not_D}(q_x\ M_1 \ldots M_{i-1}\ M_{i+1} \ldots M_n)$. However, most of those M_i, which at compile-time are variables, will be instantiated at run-time: therefore it would be incorrect to compute their complement as empty. Since we cannot foresee this instantiation, we achieve clause complementation via the introduction of disequations. This is realized by the judgment $\Gamma \vdash \mathrm{Not}_\alpha(D)$. We need the following notion: a parameter $x{:}a$ is *relevant* to a predicate symbol q (denoted xR^iq) if $\Sigma(q) = A_1 \to \cdots \to A_n \to \mathbf{o}$ and for some $1 \le i \le n$ the target type of A_i is a.

$$\frac{}{\Gamma \vdash \mathrm{Not}_\alpha(\top_q) = \bigwedge_{x \in dom(\Gamma)} (\bigwedge_{xR^iq} \mathrm{Not}^i_x(\top_q))}\,\mathrm{Not}_\alpha\top$$

$$\frac{\mathrm{Not_G}(G) = G'}{\Gamma \vdash \mathrm{Not}_\alpha(Q \leftarrow G) = (\bigwedge_{x \in dom(\Gamma)} (\bigwedge_{xR^iq} \mathrm{Not}^i_x(Q))) \wedge (\neg Q \leftarrow G')}\,\mathrm{Not}_\alpha \leftarrow$$

Both rules refer to an auxiliary judgment $\mathrm{Not}^i_x(D)$:

$$\frac{\Sigma(q) = A_1 \to \cdots \to A_n \to \mathbf{o} \qquad \cdot \vdash sh(x, A_i) = e_x}{\mathrm{Not}^i_x(\top_q) = \forall Z_1\!:\!A_1.\ \ldots \forall Z_n\!:\!A_n.\ \neg(q\ \overline{Z^i_{e_x}}) \leftarrow \top}\,\mathrm{Not}^i_x\top$$

$$\frac{\Sigma(q) = A_1 \to \cdots \to A_n \to \mathbf{o} \qquad \cdot \vdash sh(x, A_i) = e_x}{\mathrm{Not}^i_x(q\ \overline{M_n}) = \bigwedge_{1 \le j \le n, j \ne i} (\forall Z_1\!:\!A_1.\ \ldots \forall Z_n\!:\!A_n.\ \neg(q\ \overline{Z^i_{e_x}}) \leftarrow M_j \doteq Z_j)}\,\mathrm{Not}^i_x\mathrm{At}$$

The idea is to:

- Pivot on $x{:}a \in \Gamma$.
- Locate a type A_i such that $x{:}a$ is relevant to q at i.
- Complement D w.r.t. x and i.
- Repeat for every A_i and for every x.

The rest of the rules for $\Gamma \vdash \mathrm{Not}_\alpha(D)$ are completely analogous to the ones for $\mathrm{Not}_\mathrm{D}(D)$ and are omitted. Both simply recur on the program respecting the duality of conjunction and disjunction w.r.t. negation. Notice the different treatment of the trivial clause $\top$ by rules $\mathrm{Not}_\mathrm{D}\top$ and $\mathrm{Not}_\alpha\top$: if no parameter has been assumed, then $\top$ truly stands for the empty predicate definition and its complement is the universal definition $\bot$. If, on the other hand Γ is *not* empty, it means that $\top_q$ has been introduced during the $\top$-normalization preprocessing phase and has been localized to the predicate q. The rule $\mathrm{Not}^i_x\top$ allows to build a new negative assumption w.r.t. q, x, i in case $\top_q$ is the only dynamic definition of q. As $\top_q$ carries no information at all concerning q, the most general negative assumption is added; the notation $\overline{Z^i_{e_x}}$ abridges $Z_1 \ldots Z_{i-1}\ e_x\ Z_{i+1} \ldots Z_n$, where the Z's are fresh logic variables and e_x is a term built prefixing a parameter x by an appropriate number of lambda's, according to the type of its position; this is specified by the $\Gamma \vdash sh(x, A)$ judgment, omitted here (but see it in action in Example 1).

Now that we have discussed how to perform clause, assumption and goal complementation, we synchronize them together in a phase we call *augmentation*, which simply inserts the correct assumption complementation into a goal and in turn into a clause. This is achieved by a judgment $\Gamma; \mathcal{D} \vdash_\Phi aug_\mathrm{D}(D)$, again omitted here for reasons of space.

Example 1. Consider the `copy` clause on λ-terms:

$$\begin{aligned} &cplam : copy\ (lam\ E)\ (lam\ F) \\ &\qquad \leftarrow (\forall x{:}exp.\, copy\ x\ x \rightarrow copy\ (E\ x)\ (F\ x)). \end{aligned}$$

The augmentation procedure collects $x{:}exp; copy\ x\ x$ and calls $x{:}exp \vdash \mathrm{Not}_\alpha(copy\ x\ x)$. First $\mathrm{Not}^x_1(copy\ x\ x) = (\forall E'{:}exp.\, \neg copy\ E'\ x \leftarrow x \doteq\!\!\!\neq E')$, secondly $\mathrm{Not}^x_2(copy\ x\ x) = (\forall F'{:}exp.\, \neg copy\ x\ F' \leftarrow x \doteq\!\!\!\neq F')$, yielding:

$$\begin{aligned} &aug_\mathrm{D}(cplam) : copy\ (lam\ E)\ (lam\ F) \\ &\qquad \leftarrow (\forall x{:}exp. \\ &\qquad\quad (\forall E'{:}exp.\, \neg copy\ E'\ x \leftarrow x \doteq\!\!\!\neq E') \wedge \\ &\qquad\quad (\forall F'{:}exp.\, \neg copy\ x\ F' \leftarrow x \doteq\!\!\!\neq F') \rightarrow \\ &\qquad\qquad (copy\ x\ x \rightarrow copy\ (E\ x)\ (F\ x))). \end{aligned}$$

Let us see how rule $\mathrm{Not}^i_x\top$ enters the picture; recall the normalized `linxlm` clause. From $\cdot \vdash sh(y, exp \rightarrow exp) = \lambda x\,.\,y$ we have $\mathrm{Not}^y_1(\top_{linx}) = \neg linx\ (\lambda x\,.\,y)$:

$$\begin{aligned} &aug_\mathrm{D}(linxlam) : linx\ (\lambda x\,.\,lam(\lambda y\,.\,E\ x\ y)) \\ &\qquad \leftarrow (\forall y{:}exp.\, \neg linx\ (\lambda x\,.\,y) \rightarrow linx\ (\lambda x\,.\,E\ x\ y)). \end{aligned}$$

Let us apply the complement algorithm to the `linx` predicate definition:

$$
\begin{aligned}
&\mathrm{Not_D}(def(linx)) = \\
&\mathrm{Not_D}(linxx) \vee \mathrm{Not_D}(linxap1) \vee \mathrm{Not_D}(linxap2) \vee \mathrm{Not_D}(linxlm) = \\
&\quad (\neg linx(\lambda x\,.\,app\ (E_1\ x)\ (E_2\ x)) \wedge \neg linx(\lambda x\,.\,lam(\lambda y\,.\,(E\ x\ y)))) \vee \\
&\quad (\neg linx(\lambda x\,.\,x) \wedge \neg linx(\lambda x\,.\,lam(\lambda y\,.\,(E\ x\ y))) \\
&\quad \wedge \neg linx(\lambda x\,.\,app\ (E_1\ x)\ (E_2\ x)) \leftarrow strict(\lambda x\,.\,E_2\ x) \\
&\quad \wedge \neg linx(\lambda x\,.\,app\ (E_1\ x)\ (E_2\ x)) \leftarrow \neg linx(\lambda x\,.\,E_1\ x)) \vee \\
&\quad (\neg linx(\lambda x\,.\,x) \wedge \neg linx(\lambda x\,.\,lam(\lambda y\,.\,(E\ x\ y))) \\
&\quad \wedge \neg linx(\lambda x\,.\,app\ (E_1\ x)\ (E_2\ x)) \leftarrow strict(\lambda x\,.\,E_1\ x) \\
&\quad \wedge \neg linx(\lambda x\,.\,app\ (E_1\ x)\ (E_2\ x)) \leftarrow \neg linx(\lambda x\,.\,E_2\ x)) \vee \\
&\quad (\neg linx(\lambda x\,.\,x) \wedge \neg linx(\lambda x\,.\,app\ (E_1\ x)\ (E_2\ x)) \\
&\quad \wedge \neg linx(\lambda x\,.\,lam(\lambda y\,.\,(E\ x\ y))) \leftarrow (\forall y{:}exp.\,\neg linx(\lambda x\,.\,y) \rightarrow \neg linx(\lambda x\,.\,E\ x\ y))).
\end{aligned}
$$

The `strict` predicate is simply the complement of the `vac` predicate previously introduced. Again, these annotations can be internalized in the strict type theory described in [14].

We can now establish exclusivity and exhaustivity of clause complementation. Let $\mathrm{Not_D}(\mathcal{P}) = \mathcal{P}^-$:

Theorem 1 (Exclusivity). *For every run-time context $\Gamma;\mathcal{D}$ instance of a schema $\mathcal{S}$ extracted from an augmented program $\mathcal{P}$:*

1. *It is not the case that $\Gamma;\mathcal{D} \vdash_{\mathcal{P}} G$ and $\Gamma;\mathcal{D} \vdash_{\mathcal{P}^-} \mathrm{Not_G}(G)$.*
2. *It is not the case that $\Gamma;\mathcal{D} \vdash_{\mathcal{P}} (\mathcal{P} \wedge \mathcal{D}) \gg Q$ and $\Gamma;\mathcal{D} \vdash_{\mathcal{P}^-} (\mathcal{P}^- \wedge \mathcal{D}) \gg \neg Q$.*

Proof. (Sketch) By mutual induction on the structure of the derivation of $\Gamma;\mathcal{D} \vdash_{\mathcal{P}} G$ and $\Gamma;\mathcal{D} \vdash_{\mathcal{P}} (\mathcal{P} \wedge \mathcal{D}) \gg Q$. The proof goes through as there is no 'bad' interaction between the static and dynamic definition of a predicate; namely there is no overlap between a clause from $\mathcal{P}$ and from $\mathcal{D}$ since in every atomic assumption there must be an occurrence of an eigenvariable *and* every corresponding term in a program clause head must start with a constructor. If both clauses are dynamic, it holds because an appropriate disequation is present; this approximates what happens in the static case, which is based on term exclusivity.

The denial system comes in handy in the following proof.

Theorem 2 (Exhaustivity). *For every substitution θ, σ and run-time context $\Gamma;[\theta]\mathcal{D}$ instance of a schema $\mathcal{S}$ extracted from an augmented program $\mathcal{P}$:*

1. *If for all θ $\Gamma;[\theta]\mathcal{D} \nvdash_{\mathcal{P}} [\theta]G$, then there is a σ such that $\Gamma;[\sigma]\mathcal{D} \vdash_{\mathcal{P}^-} [\sigma]\mathrm{Not_G}(G)$.*

2.1 *If, for all θ $\Gamma;[\theta]\mathcal{D} \nvdash_{\mathcal{P}} [\theta]\mathcal{P} \gg [\theta]Q$, then here is a σ such that $\Gamma;[\sigma]\mathcal{D} \vdash_{\mathcal{P}^-} [\sigma]\mathrm{Not_D}(D) \gg [\sigma]\neg Q$.*

2.2 *If, for all θ $\Gamma;[\theta]\mathcal{D} \nvdash_{\mathcal{P}} [\theta]\mathcal{D} \gg [\theta]Q$, then here is a σ such that $\Gamma;[\sigma]\mathcal{D} \vdash_{\mathcal{P}^-} [\sigma]\mathrm{Not}_\alpha(D) \gg [\sigma]\neg Q$.*

The proof is by mutual induction on the structure of the given derivations. As a corollary, we are guaranteed that clause complementation satisfies the boolean rules of negation.

Finally, we show how to eliminate from clauses the '$\vee$' operator stemming from the complementation of conjunctions, while preserving provability; this will recover uniformity in proof-search.

The key observation is that in this context '$\vee$' can be restricted to a program constructor *inside* a predicate definition; therefore it can be eliminated by simulating unification in the definition, that is $(Q_1 \leftarrow G_1) \vee (Q_2 \leftarrow G_2) \equiv \theta(Q_1 \leftarrow G_1 \wedge G_2)$, where $\theta = mgu(Q_1, Q_2)$.

However, the (strict) higher-order unification problem is quite complex, even more so due to the mixed quantifier structure of HHF; since we have already parameter (dis)equations introduced by the augmentation procedure, as well as variable-variable (dis)equations stemming from left-linearization, we first compile clauses in an intermediate language which keeps the unification problems explicit and then we perform constraint simplification as in *Twelf*. Continuing with our example and simplifying the constraints:

$$\begin{aligned}
&\mathrm{Not_D}(linxx) \vee \mathrm{Not_D}(linxap1) = \\
&\quad \neg linx(\lambda x\,.\,app\ (E_1\ x)\ (E_2\ x)) \leftarrow strict(\lambda x\,.\,E_2\ x) \wedge \\
&\quad \neg linx(\lambda x\,.\,app\ (E_1\ x)\ (E_2\ x)) \leftarrow \neg linx(\lambda x\,.\,E_1\ x) \wedge \\
&\quad \neg linx(\lambda x\,.\,lam(\lambda y\,.\,E\ x\ y)).
\end{aligned}$$

The final definition of $\neg$`linear` and in turn $\neg$`linx` is:

$$\begin{aligned}
\neg linapp &: \neg linear(app\ E_1\ E_2) \\
&\qquad \leftarrow \neg linear(E_1) \vee \neg linear(E_2). \\
\neg linlam1 &: \neg linear(lam(\lambda x\,.\,E\ x)) \\
&\qquad \leftarrow \neg linx(\lambda x\,.\,E\ x) \\
&\qquad\ \vee (\forall y{:}exp.\,(\neg linx(\lambda x\,.\,y) \wedge linear(y)) \rightarrow \neg linear(E\ y)). \\
\neg linxap0 &: \neg linx(\lambda x\,.\,app\ (E_1\ x)\ (E_2\ x)) \leftarrow strict(\lambda x\,.\,E_1\ x) \wedge strict(\lambda x\,.\,E_2\ x). \\
\neg linxap1 &: \neg linx(\lambda x\,.\,app\ (E_1\ x)\ (E_2\ x)) \leftarrow \neg linx(\lambda x\,.\,E_1\ x) \wedge strict(\lambda x\,.\,E_2\ x). \\
\neg linxap2 &: \neg linx(\lambda x\,.\,app\ (E_1\ x)\ (E_2\ x)) \leftarrow \neg linx(\lambda x\,.\,E_2\ x) \wedge strict(\lambda x\,.\,E_1\ x). \\
\neg linxap3 &: \neg linx(\lambda x\,.\,app\ (E_1\ x)\ (E_2\ x)) \leftarrow \neg linx(\lambda x\,.\,E_1\ x) \wedge \neg linx(\lambda x\,.\,E_2\ x). \\
\neg linxlm &: \neg linx(\lambda x\,.\,lam(\lambda y\,.\,E\ x\ y)) \\
&\qquad \leftarrow (\forall y{:}exp.\,\neg linx(\lambda x\,.\,y) \rightarrow \neg linx(\lambda x\,.\,E\ x\ y)).
\end{aligned}$$

5 Conclusions and Future Work

We have presented elimination of negation in a fragment of higher-order HHF; our next task is to overcome some of the current restrictions, to begin with the extension to *any* order, which requires a more refined notion of context. The issue of local variables is instead more challenging. The proposal in [1] is not satisfactory and robust

enough to carry over to logical frameworks with intensional universal quantification. Our approach will be again to *synthesize* a HHF definition for the clauses with local variables which during the transformations has became *extensionally* quantified. Our final goal is to achieve negation elimination in LF.

Acknowledgments

I would like to thank Frank Pfenning for his continuous help and guidance. The notion of context schema is inspired by Schürmann's treatment of analogous material in [18].

References

[1] D. P. A. Brogi, P. Mancarella and F. Turini. Universal quantification by case analysis. In *Proc. ECAI-90*, pages 111–116, 1990.

[2] K. Apt and R. Bol. Logic programming and negation. *Journal of Logic Programming*, 19/20:9–72, May/July 1994.

[3] R. Barbuti, P. Mancarella, D. Pedreschi, and F. Turini. A transformational approach to negation in logic programming. *Journal of Logic Programming*, 8:201–228, 1990.

[4] A. Bonner. Hypothetical reasoning with intuitionistic logic. In R. Demolombe and T. Imielinski, editors, *Non-Standard Queries and Answers*, volume 306 of *Studies in Logic and Computation*, pages 187–219. Oxford University Press, 1994.

[5] K. L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Databases*, pages 293–322. Plenum Press, New York, 1978.

[6] D. M. Gabbay. N-Prolog: An extension of Prolog with hypothetical implications II. Logical foundations and negation as failure. *Journal of Logic Programming*, 2(4):251–283, Dec. 1985.

[7] L. Giordano and N. Olivetti. Negation as failure and embedded implication. *Journal of Logic Programming*, 36(2):91–147, August 1998.

[8] J. Harland. *On Hereditary Harrop Formulae as a Basis for Logic Programming*. PhD thesis, Edinburgh, Jan. 1991.

[9] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, Jan. 1993.

[10] J.-L. Lassez and K. Marriot. Explicit representation of terms defined by counter examples. *Journal of Automated Reasoning*, 3(3):301–318, Sept. 1987.

[11] R. McDowell and D. Miller. A logic for reasoning with higher-order abstract syntax: An extended abstract. In G. Winskel, editor, *Proceedings of the Twelfth Annual Symposium on Logic in Computer Science*, pages 434–445, Warsaw, Poland, June 1997.

[12] D. Miller, G. Nadathur, F. Pfenning, and A. Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51:125–157, 1991.

[13] A. Momigliano. *Elimination of Negation in a Logical Framework*. PhD thesis, Carnegie Mellon University, 2000. Forthcoming.

[14] A. Momigliano and F. Pfenning. The relative complement problem for higher-order patterns. In D. D. Schreye, editor, *Proceedings of the 1999 International Conference on Logic Programming (ICLP'99)*, pages 389–395, La Cruces, New Mexico, 1999. MIT Press.

[15] G. Nadathur and D. Miller. An overview of λProlog. In K. A. Bowen and R. A. Kowalski, editors, *Fifth International Logic Programming Conference*, pages 810–827, Seattle, Washington, Aug. 1988. MIT Press.

[16] F. Pfenning. Logical frameworks. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*. Elsevier Science Publishers, 2000. In preparation.

[17] T. Sato and H. Tamaki. Transformational logic program synthesis. In *International Conference on Fifth Generation Computer Systems*, 1984.

[18] C. Schürmann. *Automating the Meta-Theory of Deductive Systems*. PhD thesis, Carnegie-Mellon University, 2000. forthcoming.

[19] C. Schürmann and F. Pfenning. Automated theorem proving in a simple meta-logic for LF. In C. Kirchner and H. Kirchner, editors, *Proceedings of the 15th International Conference on Automated Deduction (CADE-15)*, pages 286–300, Lindau, Germany, July 1998. Springer-Verlag LNCS 1421.

Discreet Games, Light Affine Logic and PTIME Computation

A.S. Murawski* and C.-H.L. Ong**

Oxford University Computing Laboratory
Wolfson Building, Parks Rd, Oxford OX1 3QD, UK
{andrzej,Luke.Ong}@comlab.ox.ac.uk

Abstract. This paper introduces a model of IMLAL, the intuitionistic multiplicative $(\otimes \multimap \S\,!)$-fragment of Light Affine Logic, based on games and *discreet strategies*. We define a generalized notion of *threads*, so that a play of a game (of depth k) may be regarded as a number of interwoven threads (of depths ranging from 1 to k). To constrain the way threads communicate with each other, we organize them into *networks* at each depth (up to k), in accord with a protocol:

- A network comprises an O-thread (which can only be created by O) and finitely many P-threads (which can only be created by P).
- A network whose O-thread arises from a $!$-game can have at most one P-thread which must also arise from a $!$-game.
- No thread can belong to more than one network.
- Only O can switch between networks, and only P can switch between threads within the same network.

Strategies that comply with the protocol are called *discreet*, and they give rise to a fully complete model of IMLAL. Since IMLAL has a polytime cut-elimination procedure, the model gives a basis for a denotational-semantic characterization of PTIME.

Keywords: Game Semantics, Linear Logic, Complexity, PTIME.

1 Introduction

Light Linear Logic (LLL) [4] has a polytime cut-elimination procedure and can encode all polytime numeric functions. In Girard's words, it is an "intrinsically polytime system" whose proofs may be regarded as (representations of) polytime algorithms. An intuitionistic affine variant of the Logic has recently been introduced by Asperti [3]. The system, called IMLAL2 (Second-order Intuitionistic Multiplicative Light Affine Logic), is arguably simpler than LLL, and yet gives the same characterization of PTIME.

Our goal is to give a denotational characterization of PTIME by constructing a good game model of *proofs* (not just provability) of such *light logics* as LLL or IMLAL. This seems a non-trivial task: the only model of a light logic known

* On leave from Nicholas Copernicus University, Toruń, Poland.
** Webpage: http://www.comlab.ox.ac.uk/oucl/work/luke.ong.html

P. Clote and H. Schwichtenberg (Eds.): CSL 2000, LNCS 1862, pp. 427–441, 2000.

to us is one of provability, and is based on a fibred system of phase spaces [7]. The main result therein is a strong completeness theorem[1]: if a formula is valid (i.e. every valuation in any fibred phase space validates it) then it is provable. For modelling proofs, an appropriate criterion to aim for is *full completeness* [1], which is best formulated in terms of a categorical model of the logic, in which formulas are denoted by objects and proofs by maps. We say that the model $\mathbb{C}$ is *fully complete* just in case the unique functor from the relevant free category (typically the classifying category of the logic or type theory) to $\mathbb{C}$ is full. In this paper, we take the first step towards a denotational-semantic characterization of PTIME by presenting a fully complete model for the quantifier-free IMLAL.

One way to approach our model is to start from the AJM games in [2], and consider total, history-free strategies that are $\approx$-reflexive ($\approx$ is a partial equivalence relation that relates strategies which have "equivalent responses at equivalent positions"). Thus a play of a shriek game $!A$ may be viewed as a number of interwoven threads of A. The whisper game $\S A$ of light logic is a degenerate form of $!A$; a play of $\S A$ consists of only one thread of A. We introduce a generalized notion of threads at each depth i, called *i-threads*. To constrain the way threads communicate with each other, we organize threads into *networks* at each depth, in accord with a protocol. P-strategies that comply with the protocol are called *discreet* (to underline the property that P only "communicates" within a network). We can show from first principles that such strategies compose, and they give rise to a fully complete model of IMLAL. We refer the reader to the protocol (as presented in the framed box) in Section 4, and illustrate it with an example.

Example 1. Consider the two-player game of a valid sequent in Figure 1 and the twelve-move play which switches between the subgames as indicated therein (we use P and O to indicate which player has made the move). The play has two networks. Threads of the first network are contained in dashed shapes in Figure 1; they are the O-thread[2] m_1m_{12}, and two P-threads, namely, $m_2m_3m_6m_{11}$ and m_4m_5. Threads of the second network are contained in dotted shapes in the Figure; they are the O-thread m_7m_{10} and the P-thread m_8m_9. Protocol (p2) says that whenever O starts a new (O-)thread (e.g. with m_7 but not m_3), a new network is created. Protocol (p5) requires that only O may switch from one network to another existing network (e.g. from m_{10} to m_{11}), but only P may switch from one thread to another existing thread within the same network (e.g. from m_5 to m_6).

Now by replacing the formula on the right of the turnstile by $\S(B \multimap A)$, we get an invalid sequent. Consider the play that moves between the subgames, following exactly the same pattern of the preceding play. Note that the corresponding m_8 violates protocol (p5) as it is a case of P attempting to switch back to the first network.

[1] The result is for a slight variant of LLL that has a $\S$-operator which is not self-dual.
[2] For the purpose here, an O-thread is just a thread beginning with an O-move.

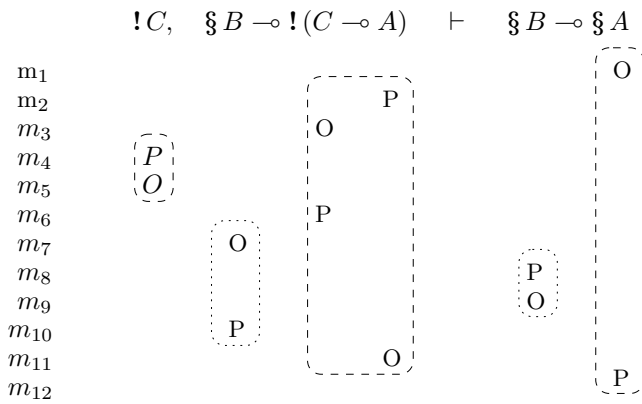


Fig. 1. Networks of (threads of) a play

To the best of our knowledge, the construction presented here is the first model of a system of light logic proofs. Our main contribution is a semantic analysis of the two crucial connectives **!** and **§**. We could have worked on a model for LLL instead but decided on the intuitionistic affine system because it is a simpler (but perfectly adequate) setting to explain our ideas. We believe we can extend discreet games to model multiplicative LLL without difficulty: games for the classical system (i.e. extended by involutive negation) can be obtained by admitting positions that begin with P-moves; weakening can be invalidated by either introducing *fairness* [5] at the level of positions or *exhaustion* [8] at the level of strategies.

2 IMLAL

Intuitionistic Multiplicative Light Affine Logic (IMLAL) formulas are generated from atoms $a, b, c, \cdots$ by the connectives $\otimes$, $\multimap$, § (read "whisper") and **!** (read "shriek"). Affine here means that the weakening rule is valid. We let Γ, Δ range over finite sequences of IMLAL formulas. The valid IMLAL sequents are defined by the rules in Figure 2. The two main results are:

Theorem 1 (Girard, Asperti).

1. *Cut elimination in a proof p of $\Gamma \vdash A$ can be done in time proportional to $|p|^{2^d}$ where d is the depth of $\bigotimes \Gamma \multimap A$ and $|p|$ is the size of p.*
2. *All PTIME numeric functions can be encoded in IMLAL2, the second-order extension of IMLAL.* □

Unfortunately we do not have the space to say anything more about the Theorem other than to direct the reader to the main references [4,3,9] for proofs and examples. The following properties of the modalities **!** and §, which are the essence of light logics, are worth emphasizing:

$$(\mathbf{var})\quad a \vdash a \qquad (\mathbf{exch})\quad \frac{\Gamma, A, B, \Delta \vdash C}{\Gamma, B, A, \Delta \vdash C}$$

$$(\mathbf{wk})\quad \frac{\Gamma \vdash B}{\Gamma, A \vdash B} \qquad (\mathbf{contr})\quad \frac{!A, !A, \Gamma \vdash C}{!A, \Gamma \vdash C}$$

$$(\otimes\mathbf{\text{-}l})\quad \frac{A, B, \Gamma \vdash C}{A \otimes B, \Gamma \vdash C} \qquad (\otimes\mathbf{\text{-}r})\quad \frac{\Gamma \vdash A \qquad \Delta \vdash B}{\Gamma, \Delta \vdash A \otimes B}$$

$$(\multimap\mathbf{\text{-}l})\quad \frac{\Gamma \vdash A \qquad B, \Delta \vdash C}{A \multimap B, \Gamma, \Delta \vdash C} \qquad (\multimap\mathbf{\text{-}r})\quad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B}$$

$$(!_0)\quad \frac{\vdash A}{\vdash !A} \qquad (!)\quad \frac{A \vdash B}{!A \vdash !B}$$

$$(\S_0)\quad \frac{\vdash A}{\vdash \S A} \qquad (\mathbf{cut})\quad \frac{\Gamma \vdash A \qquad \Delta, A \vdash B}{\Gamma, \Delta \vdash B}$$

$$(\S)\quad \frac{A_1, \cdots, A_k, B_1, \cdots, B_l \vdash C \quad k+l>0}{!A_1, \cdots, !A_k, \S B_1, \cdots, \S B_l \vdash \S C}$$

Fig. 2. The rules defining valid IMLAL sequents

1. $!$ is *not* a comonad: $!A \not\vdash A$ and $!A \not\vdash !!A$, but we have *duplication* $!A \vdash !A \otimes !A$
2. $\S$ is a degenerate or neutral form of $!$ i.e. we have $!A \vdash \S A$
3. $\S A \otimes \S B \vdash \S(A \otimes B)$ but $!A \otimes !B \not\vdash !(A \otimes B)$.

3 Games and Strategies

We consider two-player games between P (Proponent) and O (Opponent). Every play is started by O (this paper is concerned only with the intuitionistic fragment), and thereafter it alternates between P and O. Formally a ***game*** G is a three-tuple $\langle M_G, \lambda_G, P_G \rangle$ where

- M_G is a set of moves
- $\lambda_G : M_G \longrightarrow \{O, P\}$ partitions moves into those that O can make or *O-moves*, and those that P can make or *P-moves* (we will write M_G^O, M_G^P for the set of O-moves and P-moves of G respectively)
- P_G is a prefix-closed set of finite alternating sequences of moves from M_G, each beginning with an O-move; we call elements of P_G *positions* or *plays*.

For example $\langle \varnothing, \varnothing, \{\epsilon\} \rangle$ (where ϵ is the empty sequence) is a game, which we call the *empty game*. We interpret atomic formulas a as single-move games G_a, which we also call *atomic*, defined as $G_a = \langle \{a\}, \{(a, O)\}, \{\epsilon, a\} \rangle$. In a game context, a is called a ***token***. In the following we shall abuse notation and often write G_a simply as a when it is clear from the context what we mean (e.g. we abbreviate $G_a \multimap G_a$ to $a \multimap a$).

We construct new games from old using standard game constructions. We write $s \upharpoonright A$ to mean the subsequence of s consisting only of moves from A, and define $\overline{P} = O$ and $\overline{O} = P$. For a game G, we write $M_G^{\circledast}$ to mean the set of finite alternating sequences of moves from M_G. The first two, ***tensor games*** $A \otimes B$ and ***linear function space games*** $A \multimap B$, are standard. For $\circledcirc = \otimes$ and $\multimap$, we have

$$M_{A \circledcirc B} = M_A + M_B$$
$$P_{A \circledcirc B} = \{\, s \in M^{\circledast}_{A \circledcirc B} \mid s \upharpoonright A \in P_A,\ s \upharpoonright B \in P_B \,\}$$

where $\lambda_{A \otimes B}$ is defined to be the canonical map $[\lambda_A, \lambda_B] : M_A + M_B \longrightarrow \{P, O\}$, and $\lambda_{A \multimap B} = [\overline{\lambda_A}, \lambda_B]$. Note that it is a consequence of the definition that every $s \in P_{A \otimes B}$ satisfies the *O-Switching Condition*: for each pair of consecutive moves mm' in s, if m and m' are from different components (i.e. one is from A the other from B), then m' is an O-move. Similarly it follows that every $s \in P_{A \multimap B}$ satisfies the *P-Switching Condition* i.e. only P can switch component.

The next two constructions, which we call ***box constructions***, are related. The idea is that a play of a shriek game $!\,A$ consists of a number of interwoven "threads"[3] (or plays) of A, each is tagged explicitly by a number. The whisper game $\S\, A$ is a degenerate form of $!\,A$ in the sense that a play consists of just one thread, which is tagged by $\star$.

Shriek games $!\,A$	***Whisper games*** $\S\, A$
$M_{!\,A} = M_A \times \mathbb{N}$	$M_{\S\, A} = M_A \times \{\star\}$
$\lambda_{!\,A}(m, i) = \lambda_A(m)$	$\lambda_{\S\, A}(m, \star) = \lambda_A(m)$
$P_{!\,A} = \{s \in M^{\circledast}_{!\,A} \mid \forall i \in \mathbb{N}. s \upharpoonright i \in P_A\}$	$P_{\S\, A} = \{s \in M^{\circledast}_{\S\, A} \mid \pi_1^*(s) \in P_A\}$

where $s \upharpoonright i$ is the sequence of A-moves obtained first by projecting s onto the subsequence consisting of pairs whose second component is i, and then by taking the respective first projection of each pair; and where $\pi_1^*(s)$ is the sequence of A-moves obtained from s by taking the respective first projection of each pair.

A *deterministic P-strategy*, or simply ***strategy***, for a game G is a non-empty, prefix-closed subset σ of P_G satisfying: (i) for any even-length s, if $s \in \sigma$ and $sm \in P_G$ then $sm \in \sigma$, and (ii) (*determinacy*) if even-length sm and sm' are both in σ, then $m = m'$. We say that σ is *history-free* if there is a partial function $f : M_G^O \rightharpoonup M_G^P$ such that for any odd-length $sm \in \sigma$, we have $smm' \in \sigma$ if and only if $f(m)$ is defined and equal to m'; we write $\sigma = \sigma_f$ just in case f is the least such function. Further it is said to be ***injective history-free*** if the least such f is injective. If for every odd-length $s \in \sigma$, there is some m such that $sm \in \sigma$, we say that σ is ***total***. For any games A_1, A_2 and A_3 we define $\mathcal{L}(A_1, A_2, A_3)$ to be the set of finite sequences s of moves from $M_{A_1} + M_{A_2} + M_{A_3}$ such that for any pair of consecutive moves mm' in s, if $m \in M_{A_i}$ and $m' \in M_{A_j}$ then $|i - j| \leqslant 1$. (We call $\mathcal{L}(A_1, A_2, A_3)$ the set of ***interaction sequences*** over (A_1, A_2, A_3).) Take strategies σ and τ for games $A \multimap B$ and $B \multimap C$ respectively. We define

[3] In Section 4 we give a formal definition of a generalized notion of threads.

the composite $\sigma \,;\, \tau$ of σ and τ as:

$$\sigma \,;\, \tau = \{\, s \restriction (A, C) : s \in \mathcal{L}(A, B, C) \wedge s \restriction (A, B) \in \sigma \wedge s \restriction (B, C) \in \tau \,\}.$$

This is the standard notion of composition of strategies. Games and strategies (maps from A to B are strategies of $A \multimap B$) form a symmetric monoidal closed category; the category whose maps are injective history-free strategies forms a subcategory. All this is standard (see e.g. [1,8]).

Important Notation. The moves of a tensor game are just the disjoint union of the moves of the respective component games, similarly for function space games. For technical convenience, we fix representations of moves specific to the two constructions

$$\begin{array}{rcl} M_{A \otimes B} & = & M_A \times \{\, l \,\} \;\cup\; M_B \times \{\, r \,\} \\ M_{A \multimap B} & = & M_A \times \{\, L \,\} \;\cup\; M_B \times \{\, R \,\} \end{array}$$

We call ***free games*** those that are constructed from atomic games. *We emphasize that from now on, by games we shall mean free games.* A move of a free game has the form

$$m = ((\cdots((a, i_n), i_{n-1})\cdots), i_1)$$

where a is a token and each i_j ranges over $\mathbb{N} \cup \{\star, l, r, L, R\}$. For convenience, we shall write the move as a pair $(a, i_n \cdots i_1)$, and call the second component its *occurrence*. Let $i_{r_d} \cdots i_{r_1}$ be the subsequence of $i_n \cdots i_1$ consisting only of numbers and $\star$. We define the ***depth*** of the move m to be d, the *index at depth* j, or j***-index***, to be the *sequence* $i_{r_j} i_{r_j - 1} \cdots i_3 i_2 i_1$, and the ***token*** to be a. For example, the 1-index, 2-index and 3-index of the move $(a, 2r\star Ll33R)$ are $3R, 33R$ and $\star Ll33R$ respectively. We say that a subgame A of a game G occurs at depth i if A is in the scope of i box constructors. A game is said to have ***depth*** k just in case k is the maximum depth of all its subgames. (It is straightforward to show that k is equal to the maximum depth of its moves.) We say that a strategy σ for a free game is ***token-reflecting*** if for any even-length $smm' \in \sigma$, m and m' have the same token.

Example 2. Consider the game $\mathbf{!}\,(\mathbf{!}\,a \otimes \mathbf{!}\,b) \multimap \mathbf{!}\,\S\,a$ (which, *qua* formula, is provable); $(a, \star 3R)(a, 2l5L)$ and $(a, \star 3R)(b, 1r4L)$ are positions of the game. Observe that the occurrence of a move may be read as a path by which its token can be reached from outside-in.

It is straightforward to show that IMLAL proofs are denoted by token-reflecting, injective history-free, total strategies. Indeed these properties are enough to characterize denotations of proofs of the $(\otimes, \multimap)$-fragment of Intuitionistic Multiplicative Affine Logic, which we abbreviate to IMAL. (Valid IMAL sequents are those defined by the first eight rules in Figure 2.)

Theorem 2 (Full Completeness). *For any free game given by an IMAL-sequent $\Gamma \vdash A$, and for any* ***IMAL-winning*** *(i.e. token-reflecting, injective history-free, total) strategy σ for the game, there is a derivation of $\Gamma \vdash A$ whose denotation is σ.* □

(A proof of the Theorem can be extracted from [8].)

However, to characterize IMLAL proofs, among other conditions, we require strategies to act uniformly on threads, as the next example illustrates.

Example 3. Consider the game $!\,a \otimes !\,a \multimap !\,a$; the strategy defined by the function f: for $n \in \mathbb{N}$

$$f \;:\; \begin{cases} (a,(2n)R) \mapsto (a,nlL) \\ (a,(2n+1)R) \mapsto (a,nrL) \end{cases}$$

is not the denotation of any proof of the corresponding sequent (there are in fact only two proofs).

Thread uniformity and $\approx$-reflexivity. In order to cut the strategies down to size, we use a partial equivalence relation $\approx$ over strategies introduced in [2]. First we define a $\approx$-***game*** G to be a four-tuple $\langle M_G, \lambda_G, P_G, \approx_G \rangle$ where $\langle M_G, \lambda_G, P_G \rangle$ is a game, and $\approx_G$ is an equivalence relation on P_G satisfying:

- $sa \approx_G tb \;\Rightarrow\; s \approx_G t$
- $s \approx_G t \;\Rightarrow\; \lambda^*_G(s) = \lambda^*_G(t)$
- $s \approx_G t \wedge sa \in P_G \;\Rightarrow\; \exists b \in M_G.tb \in P_G \wedge sa \approx_G tb.$

We extend the four game constructions introduced earlier to their respective $\approx$-game constructions. For $\circledcirc = \otimes$ or $\multimap$, we have $s \approx_{A \circledcirc B} t$ just in case $s \upharpoonright A \approx_A t \upharpoonright A$, $s \upharpoonright B \approx_B t \upharpoonright B$ and $\pi_2(s_i) = \pi_2(t_i)$, for each i, where s_i denotes the i-th move of s. For whisper games $\S\, A$, we have $s \approx_{\S\, A} t$ if $\pi^*_1(s) \approx_A \pi^*_1(t)$. For shriek games $!\, A$, we have $s \approx_{!\, A} t$ if

$$\exists \alpha \in S(\mathbb{N}).\forall i \in \mathbb{N}.s \upharpoonright i \approx_A t \upharpoonright \alpha(i) \;\wedge\; \alpha^*(\pi^*_2(s)) = \pi^*_2(t)$$

where $S(\mathbb{N})$ is the collection of permutations of $\mathbb{N}$. The equivalence relation $\approx_G$ extends to a *partial* equivalence relation over strategies of G: $\sigma \approx_G \tau$ holds just in case for all $s \in \sigma$, $t \in \tau$, $sa \in P_G$, $tc \in P_G$, if $sa \approx_G tc$ then the following bisimulation-style properties hold:

- $sab \in \sigma \;\Rightarrow\; \exists d \in M_G.tcd \in \tau \wedge sab \approx_G tcd$
- $tcd \in \tau \;\Rightarrow\; \exists b \in M_G.sab \in \sigma \wedge sab \approx_G tcd.$

The strategies that we are interested in are those σ that are $\approx$-***reflexive*** i.e. $\sigma \approx \sigma$; we write the partial equivalence class of σ as $[\sigma]$. The intuition is that such strategies "behave equivalently at equivalent positions". For instance two threads of a shriek game $!\, A$ that have equivalent underlying A-positions, but tagged by different numbers, are equivalent. Thus we reject the strategy σ_f in Example 3: its response at even-numbered threads is different from that at equivalent but odd-numbered threads); see Example 6 for another illustration of the effect of $\approx$.

It turns out that $\approx$ is preserved by composition, so that games, with maps from A to B given by $\approx_{A \multimap B}$ partial equivalence classes of injective history-free strategies of $A \multimap B$, form a symmetric monoidal closed category (see [2]). From now on, by games we shall mean $\approx$-*reflexive free games.*

4 Network Protocol and Discreet Strategies

In this section we introduce the network protocol and consider the strategies that comply with it. First we need to formalize a generalized notion of *threads* of a game. Fix a free game G of depth k. By an *i-index* of G, we mean the i-index of some move of G (so $i \leqslant k$). *Threads of depth i*, or simply ***i-threads***, of G are named by i-indices of G. A thread named by θ is the set of moves of G whose i-index is θ. (We start counting from outside-in, so that the outermost thread is at depth 1.) An i-thread named by θ is said to be a ***P-thread*** if there are an odd number of occurrences of L in θ, otherwise it is said to be an ***O-thread***; if the leftmost symbol of the index θ is a number we say the i-thread is of **!**-***type***, otherwise the thread is of §-***type***.

Remark 1. (i) For any game G, we write $T^P_{G,i}$ (respectively $T^O_{G,i}$) for the set of P-threads (respectively O-threads) of G at depth i. We shall omit the subscript i whenever we can get away with it. For any A and B it is easy to see that

$$\begin{array}{lll} T^O_{A\multimap B} \cong T^P_A + T^O_B & T^O_{A\otimes B} \cong T^O_A + T^O_B & T_{!\,A} \cong T_A \times \mathbb{N} \\ T^P_{A\multimap B} \cong T^O_A + T^P_B & T^P_{A\otimes B} \cong T^P_A + T^P_B & T_{\S\,A} \cong T_A \end{array}$$

(ii) We shall often analyse a play s by considering its subsequence consisting of moves belonging to a given i-thread. We call that subsequence the ***i-thread of the play*** s. (Thus *i-thread of a game* is a set, and *i-thread of a play* is a sequence, of moves.)

(iii) It is straightforward to prove, by induction on the structure of the game, that any O-thread (respectively P-thread) of a play must begin with an O-move (respectively P-move).

Example 4. Consider the game $!!(a \multimap a) \multimap !!(a \multimap a)$ and the play

$$(a, R41R)\ (a, R32L)\ (a, L32L)\ (a, R92L)\ (a, L92L)\ (a, L41R)$$

which we refer to as $m_1 \cdots m_6$ in the picture (where only the respective numeric subsequences of the six occurrences are shown)

	$!!(a$	$\multimap$	$a)$	$\multimap$	$!!(a$	$\multimap$	$a)$
m_1							41
m_2			32				
m_3	32						
m_4			92				
m_5	92						
m_6					41		

The two 1-threads of the play are m_1m_6 (named by $1R$) and $m_2m_3m_4m_5$ (named by $2L$); and the three 2-threads are m_1m_6 (named by $41R$), m_2m_3 (named by $32L$) and m_4m_5 (named by $92L$).

A network protocol. To model IMLAL, we need to constrain the way threads interact or communicate with each other. A key innovation of this paper is to organize threads into *networks* in accord with a protocol. We are interested in plays that obey the following network protocol at *every* depth (up to the depth of G):

(p1) A ***network*** (at depth d) comprises an O-thread and finitely many P-threads (all at depth d).

(p2) ***Network and Thread Creation***: Only O can open a new network, and he does so whenever he starts a new O-thread; whenever P opens a new thread in response to an O-move from a network, a new P-thread is added to that network.

(p3) **!-*Network***: A network whose O-thread is of !-type (call the network a !-*network*) has at most one P-thread which must also be of !-type.

(p4) No thread can belong to more than one network.

(p5) ***Switching Condition***: Only O can switch network i.e. revisit one opened earlier or enter the *threadless universe*[a]. Only P can switch from one thread to another existing thread within the same network.

[a] At depth d, the threadless universe of G consists of moves of depth $< d$.

In the following we motivate and illustrate the protocol rules by considering the game denotations of several simple IMLAL sequents.

Example 5. (i) Take the game $a \multimap !a$. Any opening move starts an O-thread and hence a network. The only P-move of the game is from the threadless universe, and P responding with that would violate protocol (p5). There is no total strategy for the game $!!a \multimap !a$ because (p2) would be violated at depth 2; similarly for $!a \multimap a$ and $!a \multimap !!a$. The preceding analysis applies in each case where $!$ is replaced by $\S$. Thus neither of the box constructors is a comonad.

(ii) Consider the token-reflecting strategies for the respective games denoting the sequents:

$$\vdash \S a \otimes \S b \multimap \S (a \otimes b)$$
$$\nvdash \S (a \otimes b) \multimap \S a \otimes \S b$$

For the first game, it is an easy exercise to check that the positions of the token-reflecting strategy obey the protocol. For the second, there are two O-threads (named respectively by $\star lR$ and $\star rR$) and the one P-thread belongs to both the respective networks specified by the O-threads, thus violating (p4).

(iii) Consider the respective game denotations of the sequents:

$$\nvdash !a \otimes !b \multimap !(a \otimes b)$$
$$\vdash !(a \otimes b) \multimap !a \otimes !b$$

The positions $(a, l1R)\,(a, 1lL)$ and $(b, r1R)\,(b, 1rL)$ belong to the token-reflecting total strategy for the first game. The respective first moves $(a, l1R)$ and $(b, r1R)$ belong to the same O-thread of the game, and hence, the same network, but the respective second moves $(a, 1lL)$ and $(b, 1rL)$ belong to different P-threads, thus violating (p3). The second sequent is provable in IMLAL but not in LLL.

(iv) Consider the game $!!\,(a \multimap a) \multimap !!\,(a \multimap a)$ of Example 4. The play therein violates (p3): the $!$-network at depth 2 has two P-threads. However it is easy to see that the formula is provable. Indeed the game has a position that complies with the protocol, for instance

$$(a, R53R)\ (a, R97L)\ (a, L97L)\ (a, L53R).$$

For a related example, take the game $!!\,(a \multimap a) \multimap !\,\S\,(a \multimap a)$ (whose corresponding formula is provable). The play

$$(a, R{\star}1R)\ (a, R32L)\ (a, L32L)\ (a, R92L)\ (a, L92L)\ (a, L{\star}1R)$$

corresponds to the play in Example 4 but it complies with the protocol.

(v) Consider the respective token-reflecting strategies for the games denoting the following sequents:

(1) $\not\vdash\ (!\,a \multimap !\,b) \multimap !\,(a \multimap b)$
(2) $\not\vdash\ !\,(a \multimap b) \multimap (!\,a \multimap !\,b)$
(3) $\not\vdash\ (\S\,a \multimap \S\,b) \multimap \S\,(a \multimap b)$
(4) $\vdash\ \S\,(a \multimap b) \multimap (\S\,a \multimap \S\,b)$

In the first and the third games, the fourth move violates protocol (p5) as it is a case of P attempting to switch back to the original network. In the second game, protocol (p3) is violated by the fourth move as P tries to open a second $!$-thread of a $!$-network. The strategy for the fourth game complies with the protocol.

Here are some useful consequences of the network protocol.

Lemma 1. *1. Each P-move has the same depth as the preceding O-move.*
2. Each network at depth $d+1$ is embedded into a single network at d.
3. P's switching condition is a consequence of O's. □

We leave the largely straightforward proofs as an exercise.

Networked positions and discreet strategies. We are now in a position to give a formal definition of strategies that comply with the network protocol. We denote the set of threads of a game G by T_G, which partitions into T_G^O (the set of O-threads) and T_G^P (the set of P-threads), and we write $T_{G,i}$ for the set of i-threads. We formalize the network protocol with the help of two (kinds of) functions.

A *thread function* (at depth i) is a partial function $t_{G,i} : M_G \rightharpoonup T_{G,i}$ that maps a move to the i-thread in which it occurs. Whenever the game G and depth i are clear from the context, we will abbreviate $t_{G,i}(m)$ to t_m.

Network function (at depth i) $\eta_{G,i}$. We say that a partial function

$$\eta_{G,i} : T^P_{G,i} \rightharpoonup T^O_{G,i}$$

networks a position $s \in P_G$ of even length at depth i (with respect to $t_{G,i}$) just in case for each odd j, $t_{G,i}(s_j)$ is defined if and only if $t_{G,i}(s_{j+1})$ is, and if both are defined, one of the following holds (we drop the subscripts from $\eta_{G,i}$ and $t_{G,i}$ whenever we safely can):

$$\begin{array}{llll} (i) & \eta(t(s_j)) = t(s_{j+1}) & (ii) & t(s_j) = \eta(t(s_{j+1})) \\ (iii) & \eta(t(s_j)) = \eta(t(s_{j+1})) & (iv) & t(s_j) = t(s_{j+1}) \end{array}$$

(This is just to say that at depth i the P-move s_{j+1} is in the same network as the O-move s_j: there are four cases depending on whether s_j and s_{j+1} are in a P-thread or an O-thread.) In addition, for any O-thread t^O, if t^O is of $!$-type then the inverse image of t^O, written $\eta^{-1}(t^O)$, contains at most one thread which must also be of $!$-type. □

The astute reader will notice that we have not yet captured all the protocol rules. What is missing is a ***compactness condition*** which is part of (p1): $\eta^{-1}(t^O)$ is finite for every O-thread t^O.

Example 6. To see why the compactness condition is desirable, consider the (first-order) Church numerals game (see [4,3] for further details)

$$!\,(a \multimap a) \;\multimap\; \S\,(a \multimap a).$$

Let η_n be the (least) function that networks any fixed n-th Church numeral strategy (any two such n-th Church numeral strategies are $\approx$-equivalent); and let t^O be the unique O-thread of the game. It is easy to see that $\eta_n^{-1}(t^O)$ has exactly n P-threads. The "infinity numeral" strategy is ruled out by the compactness condition.

We say that a strategy σ on G is *networked at depth i* if there exists a function $\eta_{G,i}$ that networks every $s \in \sigma$ at depth i. Further we say that it is *compactly networked at depth i* if the compactness condition is satisfied. A strategy σ is said to be ***discreet*** if it is compactly networked at every depth up to the depth of G.

5 A Model of IMLAL

We are interested in token-reflecting, injective history-free, $\approx$-reflexive, total discreet strategies. To save writing, we call such strategies ***winning***. Let $\mathbb{G}$ be the category whose objects are free games, and whose maps $A \longrightarrow B$ are $\approx_{A \multimap B}$ partial equivalence classes of winning strategies of $A \multimap B$. Our first task is to check that $\mathbb{G}$, as specified by the above data, is a well-defined category – the main technical problem is to prove that total discreet strategies compose. Secondly we show that $\mathbb{G}$ is a model of IMLAL.

Total discreet strategies compose. Consider the games A, B and C and suppose u is an interaction sequence of two positions $p_1 \in P_{A \multimap B}$ and $p_2 \in P_{B \multimap C}$ which are networked by $\eta_{A \multimap B,i}$ and $\eta_{B \multimap C,i}$ respectively at a given depth i. First we would like to show that $u \upharpoonright (A, C)$ can be networked, and that all positions of the composite strategy are networked by the same network function $\eta_{A \multimap C,i}$.

Notation. Fix a network function η_G. We shall write $t' \prec_G t$ to mean $\eta_G(t) = t'$, so for each P-thread t there can be at most one O-thread t' such that t and t' are in the same network. Our argument in the following is with respect to a fixed depth i. For ease of writing, we shall drop all references to the depth and omit the subscript i from $\eta_{A \multimap C,i}$, $t_{A \multimap C,i}$ etc.

First we define the network function $\eta_{A \multimap C}$: for any $t^O \in T^O_{A \multimap C}$ and $t^P \in T^P_{A \multimap C}$, we decree that $\eta_{A \multimap C}(t^P) = t^O$ (i.e. $t^O \prec_{A \multimap C} t^P$) just in case there exist $b_1, \cdots, b_k \in T_B$ such that

$$t^O \prec_1 b_1 \prec_2 b_2 \prec_3 \cdots \prec_{k-1} b_{k-1} \prec_k b_k \prec_{k+1} t^P \tag{1}$$

where each $\prec_i \in \{\prec_{A \multimap B}, \prec_{B \multimap C}\}$, and for each $i \in \{1, \cdots, k\}$, $\prec_i \neq \prec_{i+1}$. (Each b_i in (1) above is actually the embedding image of the B-thread in $T_{A \multimap B}$ or $T_{B \multimap C}$ as appropriate – see Remark 1, but we stick to b_i by abuse of notation.) Note that if t^O and t^P are both A-threads (respectively C-threads) and $t^O \prec_{A \multimap B} t^P$ (respectively $t^O \prec_{B \multimap C} t^P$), then $t^O \prec_{A \multimap C} t^P$ which is the case of $k = 0$.

Because $\eta_{A \multimap B}$ and $\eta_{B \multimap C}$ are partial functions, we first observe that $\eta_{A \multimap C}$ is a partial function from $T^P_{A \multimap C}$ to $T^O_{A \multimap C}$.

Proposition 1. *Take any contiguous segment $m_O m_1 \cdots m_k m_P$ of u such that $m_O \in M^O_{A \multimap C}$, $m_P \in M^P_{A \multimap C}$, and for $i = 1, \cdots, k$, $m_i \in M_B$. If m_O is part of a thread then m_O and m_P are in the same network with respect to $\eta_{A \multimap C}$ i.e. writing t^O as the O-thread of m_O's network, we have either $t^O = t_{m_P}$ or $t^O \prec_{A \multimap C} t_{m_P}$.* □

Note that $\eta_{A \multimap C}$ has been defined only in terms of $\eta_{A \multimap B}$ and $\eta_{B \multimap C}$. Moreover, if σ and τ satisfy (p3), so does $\sigma \,;\, \tau$, because if

$$t^O \prec_1 b_1 \prec_2 b_2 \prec_3 \cdots \prec_{k-1} b_{k-1} \prec_k b_k \prec_{k+1} t^P$$

and t^P is a $!$-thread, then by (p3) applied alternately to σ and τ, we deduce that $b_1, \cdots, b_k$ and t^O are unique and are all $!$-threads. Thus if strategies σ and τ are networked at depth i by $\eta_{A \multimap B}$ and $\eta_{B \multimap C}$ respectively, $\sigma \,;\, \tau$ is networked by $\eta_{A \multimap C}$ at depth i. Since the preceding argument is independent of i, we can conclude:

Theorem 3. *Strategies that are networked at all depths compose.* □

By first proving the following

Proposition 2. *Suppose $\sigma : A \multimap B$ and $\tau : B \multimap C$ are networked strategies. There exists no interaction sequence u over (A, B, C) such that for some $d \geqslant 1$:*

1. *the number of (distinct) d-indices of threads occurring in* $u \upharpoonright B$ *is infinite*
2. *the number of d-indices of O-threads occurring in* $u \upharpoonright A$ *and* $u \upharpoonright C$ *is finite*
3. $u \upharpoonright (A, B) \in \sigma$ *and* $u \upharpoonright (B, C) \in \tau$. □

we can deduce that total *compactly* networked strategies compose. To summarize, we have

Theorem 4 (Compositionality). *Discrete strategies compose.* □

$\mathbb{G}$ is a model of IMLAL. For any winning strategy of $A \multimap B$ defined by the function f, and for any $\mathbb{N}$-*involution* α (i.e. α is an idempotent bijection from $\mathbb{N}$ to $\mathbb{N}$) we define a strategy of $!\,A \multimap !\,B$ given by the function $!\,(f, \alpha)$. Suppose $f(m, \delta) = (m', \delta')$ where $\delta, \delta' \in \{ L, R \}$, we define

$$!\,(f,\alpha) \ : \ ((m,i),\delta) \quad \mapsto \quad \begin{cases} ((m',i),\delta') & \text{if } \delta = \delta' \\ ((m',\alpha(i)),\delta') & \text{otherwise} \end{cases}$$

It is straightforward to show that $!\,(f, \alpha)$ defines a winning strategy, and that $\sigma_{!\,(f,\alpha)} \approx \sigma_{!\,(f,\beta)}$ for any $\mathbb{N}$-involution β. Indeed $!$ extends to a functor $\mathbb{G} \longrightarrow \mathbb{G}$. We leave the definition of the functor $\S : \mathbb{G} \longrightarrow \mathbb{G}$ as an easy exercise.

Proposition 3. *1.* $\S, ! : \mathbb{G} \longrightarrow \mathbb{G}$ *are functorial.*
2. There are canonical $\mathbb{G}$*-maps, natural in* A *and* B *as follows:*

$$\begin{array}{rccc} \mu_{A,B} : & \S\,A \otimes \S\,B & \longrightarrow & \S\,(A \otimes B) \\ \nu_A : & !\,A & \longrightarrow & \S\,A \\ \Delta_A : & !\,A & \longrightarrow & !\,A \otimes !\,A \end{array}$$

By now the reader should have no difficulty in constructing the canonical maps. By abuse of notation, we use μ also for the canonical map

$$\mu \ : \ \S\,A_1 \otimes \cdots \otimes \S\,A_n \ \longrightarrow \ \S\,(A_1 \otimes \cdots \otimes A_n).$$

As $\mathbb{G}$ is symmetric monoidal closed, and in view of the Proposition, we can say that $\mathbb{G}$ is a model of IMLAL.

6 Full Completeness

The section concerns a full completeness result:

Theorem 5 (Full Completeness). *For any winning strategy* σ *for the free game* G *given by an IMLAL sequent* $\Gamma \ \vdash \ C$*, there is a derivation* Ξ *of the sequent such that* σ *is the denotation of* Ξ. □

In the following we sketch a proof of the Theorem. First we give two useful lemmas.

Lemma 2 (Deboxing). *In the category* $\mathbb{G}$*:*

1. *For any map* $F : !A \longrightarrow !B$ *there is a unique map* $f : A \longrightarrow B$ *such that* $F = !f$.
2. *If the map* $F : !A_1 \otimes \cdots \otimes !A_m \otimes \S B_1 \otimes \cdots \otimes \S B_n \longrightarrow \S C$ *uses each* $!A_i$ *once (in the sense that the depth-1 network of any winning strategy that represents* F *comprises one 1-thread of* $!A_i$*), then there is a unique map*

$$f \ : \ A_1 \otimes \cdots \otimes A_m \otimes B_1 \otimes \cdots \otimes B_n \longrightarrow C$$

such that $F = (\underbrace{\nu \otimes \cdots \otimes \nu}_{m} \otimes \underbrace{\mathsf{id} \otimes \cdots \otimes \mathsf{id}}_{n}) \ ; \ \mu \ ; \ \S f$*, where* μ *and* ν *are the canonical maps defined in Proposition 3.* □

Let σ be a IMAL-winning strategy for a free game given by an IMAL-sequent $\Delta \vdash P \otimes Q$ where $\Delta = D_1, \cdots, D_n$.We say that σ is ***splitting*** just in case there is a partition of Δ into (Δ_1, Δ_2) and IMAL-winning strategies σ_1 and σ_2 for $\Delta_1 \vdash P$ and $\Delta_2 \vdash Q$ respectively such that $\sigma = \sigma_1 \otimes \sigma_2$.

Lemma 3 (Pivot). *If* σ *is not splitting, then there is some* $D_i = A \multimap B$ *in* Δ *(which we shall call a* ***pivot****) and IMAL-winning strategies* τ *and* υ *for* $\Theta \vdash A$ *and* $B, \Xi \vdash P \otimes Q$ *respectively, where* (Θ, Ξ) *is a partition of* $\Delta \setminus \{ D_i \}$*, such that* σ *can be defined in terms of* τ *and* υ*.* □

We prove the Theorem by induction on the size of the sequent $\Gamma \vdash C$ that defines the free game G. W.l.o.g. we assume that no formula in $\Gamma = C_1, \cdots, C_n$ is a tensor, and every C_i is σ-*reachable* in the sense that there is a position in σ that contains some C_i-move.

Step 1. Decontract at depth 1, if necessary, to get a corresponding winning strategy σ_1 for $\Gamma_1 \vdash C$ so that every formula in Γ_1 is used once by σ_1.

Step 2. It suffices to consider the following cases of C:

I. $C = P \otimes Q$
II. $C = \Box P$, a *box formula* i.e. $\Box = !$ or $\S$
III. $C = a$, an atom.

For Case I, if $P \otimes Q$ is splitting, then split it to get two smaller instances (of a winning strategy for a free game). Otherwise, we transform the sequent by adding a fresh atom as "$x \multimap -$" inside each box-formula from Γ_1 and adding a copy of "$x \multimap -$" of negative polarity inside the O-box of the corresponding network. E.g. the sequent $\S c, !(c \multimap d), \S(d \multimap e), \S e \multimap !a \otimes !b, !c \ \vdash \ \S(c \otimes a) \otimes \S b$ is transformed to

$$\S(x \multimap c), !(y \multimap (c \multimap d)), \S(z \multimap (d \multimap e)),$$
$$\S(x \multimap (y \multimap (z \multimap e))) \multimap !a \otimes !b, !(v \multimap c) \ \vdash \ \S(v \multimap c \otimes a) \otimes \S b$$

Call the new sequent $\Gamma' \vdash C'$ and the corresponding strategy σ'. Note (i) σ' is a winning strategy; (ii) the new $\multimap$-formulas added cannot be pivots as they communicate with the O-threads of their respective networks. Now consider a

second transformation to (say) $\Gamma'' \vdash C''$ which is defined by "forgetting all the boxes" and turning it into a free IMAL-game, and call the corresponding strategy σ''. Observe that if σ'' is splitting in IMAL then σ' is also splitting in IMLAL, because all formulas in Γ'' are $\multimap$-formulas or atoms. Suppose σ'' is not splitting in IMAL. Then by Lemma 3 there must be a pivot in Γ''. By construction of Γ', that pivotal $\multimap$-formula cannot occur inside a box in Γ_1, so the pivot is also a pivot for $\Gamma_1 \vdash P \otimes Q$. With the pivot, we obtain two smaller instances, to which we can apply the induction hypothesis.

For Case II, if Γ_1 does not contain a $\multimap$-formula, then every formula in it must be a box-formula; we use Lemma 2 to strip off the outermost boxes, and so obtain a smaller instance, and then apply the induction hypothesis. Otherwise, $\Box P$ is part of some network $\Box G_1, \cdots, \Box G_l$. For each $G = P, G_1, \cdots G_l$, we replace $\Box G$ by $\Box G \otimes \Box G$. E.g. the sequent $c, c \multimap \mathbf{!}\, b, \S\,(b \multimap a) \vdash \S\, a$ is transformed to

$$c, c \multimap (\mathbf{!}\, b \otimes \mathbf{!}\, b), \S\,(b \multimap a) \otimes \S\,(b \multimap a) \vdash \S\, a \otimes \S\, a$$

Let σ' be the corresponding strategy for the transformed game $\Gamma' \vdash \Box P \otimes \Box P$; σ' is not splitting (because the $\multimap$-formula in Γ' is reachable from both $\Box P$ on the right). This is an instance of the preceding case. Finally a similar (but simpler) transformation can be applied to reduce Case III to Case I.

Further work. An obvious direction is to extend the construction to IMLAL2, the second-order system. Another is to check that the model is not just fully but also *faithfully* complete. We expect to prove this with respect to an appropriate notion of equality between IMLAL proof nets. Also worth developing is a convenient syntax for IMLAL2 as a PTIME intermediate (or meta-) programming language.

Acknowledgments. We are grateful to Hanno Nickau for discussions on LLL.

References

1. Abramsky, S., Jagadeesan, R.: Games and full completeness for multiplicative linear logic. *Journal of Symbolic Logic* **59** (1994) 543–574
2. Abramsky, S., Jagadeesan, R., Malacaria, P.: Full abstraction for PCF (extended abstract). In *Proc. TACS'94*. LNCS **789** (1994) 1–15
3. Asperti, A.: Light affine logic. In *Proc. LICS'98*. IEEE Computer Society (1998)
4. Girard, J.-Y.: Light linear logic. *Information & Computation* **143** (1998) 175–204
5. Hyland, J. M. E., Ong, C.-H. L.: Fair games and full completeness for Multiplicative Linear Logic without the MIX-rule. Preprint (1993)
6. Hyland, J. M. E., Ong, C.-H. L.: On Full Abstraction for PCF: I, II & III. To appear in *Information & Computation* (2000) 130 pp
7. Kanovich, M. I., Okada, M., Scedrov, A.: Phase semantics for light linear logic. In *Proc. 13th Conf. MFPS'97*. ENTCS **6** (1997)
8. Murawski, A. S., Ong, C.-H. L.: Exhausting Strategies, Joker Games and Full Completeness for IMLL with Unit. In *Proc. 8th Conf. CTCS'99*. ENTCS **29** (1999)
9. Roversi, L.: A PTIME Completeness Proof for Light Logics. In *Proc. 13th Conf. CSL'99*. LNCS **1683** (1999)

Completeness of Higher-Order Duration Calculus *

Zhan Naijun

Lab. of Computer Science and Techonology, Institute of Software,
the Chinese Academy of Sciences, Beijing, 100080, P.R. China
ttznj@ox.ios.ac.cn

Abstract. In order to describe the real-time behaviour of programs in terms of Duration Calculus (DC), proposed by Zhou Chaochen, C.A.R. Hoare and A.P. Ravn in [3], which can specify real-time requirements of computing systems, quantifications over program variables are inevitable, e.g. to describe local variable declaration, to declare internal channel and so on. So a higher-order duration calculus (HDC) is established in [2]. This paper proves completeness of HDC on abstract domains by encoding HDC into a complete first-order two-sorted interval temporal logic (IL_2). This idea is hinted by [9]. All results shown in this paper are done under the assumption that all program variables have finite variability.

Keywords: duration calculus higher-order logic interval temporal logic completeness

1 Introduction

In order to describe the real-time behaviour of programs in terms of DC, quantifications over program variables are inevitable, e.g. to describe local variable declaration and so on. So a higher-order duration calculus is established in [2]. In [2], a real-time semantics of local variables has been demonstrated, and some real-time properties of programs have been derived using HDC.

In order to specify the behaviour of real-time programs, program variables $V_i,\ i \geq 0$ are introduced into HDC. Predicates of program variables, constants, and global variables, such as $(V < 3)$ and $(V = x)$, are taken as states. To axiomatise the finite variability of program variables, the infinite rule (ω-rule) proposed in [8] is necessary, since [5] has shown that the finite variability cannot be axiomatised by finite rules on abstract domains.

In programming languages, value passing involves past and future time, to receive an initial value from the previous statement and to pass final value to the next statement. The chop modality ";" is a contracting one, and cannot express state properties outside the current interval. Therefore, two special functions

* The work is partially supported by UNU/IIST, and done during the author stayed at UNU/IIST as a follow (July 1998 to August 1999). The work is also partially supported by the National Natural Science Foundation of China under grant No. 69873003.

P. Clote and H. Schwichtenberg (Eds.): CSL 2000, LNCS 1862, pp. 442–456, 2000.

"$\leftarrow$" and "$\rightarrow$", firstly proposed in [4], are introduced into HDC. The functions "$\leftarrow$" and "$\rightarrow$" have a domain of state terms and a co-domain of functions from the intervals to duration domain. E.g. $\overleftarrow{V}= 4$ means that in a left neighbourhood of the current interval the value of V is 4. Symmetrically, $\overrightarrow{V}= 4$ means that in a right neighbourhood of the current interval the value of V is 4. In order to axiomatise them, the neighbourhood rule is introduced in [4].

In both interval temporal logic [5] and duration calculi [3,4], symbols are divided into flexible and rigid symbols (adopting the terminology of [1,7]). Rigid symbols are intended to represent fixed, global entities. Their interpretation will be the same in all the intervals. Conversely, entities which may vary in different intervals are represented by flexible symbols. Such a distinction between two classes of symbols is common in the context of first order temporal logics [1,7].

Completeness of interval temporal logics and duration calculi not only depends on the choice of time domain, but also relies on which kind of variables are quantified. In practice, we need to choose the reals as time domain. If so, we cannot get completeness of these systems, for if they were, they would be adequate for arithmetic, which is impossible by Gödel's Theorem. Therefore, if we want to choose the reals as time domain, we can only get relative completeness of these systems. E.g. relative completeness of DC has been proved in [10]. If we only quantify over global variables, duration calculi are complete on abstract domains shown in [8]. But if we introduce quantifications over program variables into DC, since we interpret program variables as functions from time domain to duration domain, no (consistent) system is complete for this semantics because whenever we interpret the domain of quantifiers as the set of all functions from time domain to duration domain, the language will have the expressive power of second-order arithmetic. So some restrictions on program variables are needed in order to work out a complete proof system, that is, that all program variables vary finitely is assumed. If so, we can reduce HDC to IL_2 . We will illustrate it as follows:

A naive way to reduce the second order logic to the first order one is to introduce for the class of n-ary predicates, $H^n(x_1, ..., x_n)$, a new $(n + 1)$-ary predicate, $E^{n+1}(z, x_1, ..., x_n)$, which has an additional argument z, and enumerates all $H^n(x_1, ..., x_n)$. Thus,

$$\exists H^n.\phi$$

could be reduced to

$$\exists z.\phi[E^{n+1}(z, x_1, ..., x_n)/H^n(x_1, ..., x_n)]$$

Therefore the second order logic could be reduced to a first order one. Detail discussion about this encoding can be seen in [6]. However, in order to define the $(n + 1)$-ary predicate E^{n+1}, we must have the following postulates, where we assume $(n = 1)$ and drop the indices of n and $(n + 1)$ for simplicity. Firstly,

$$\exists z.E(z, x_1) \qquad \text{and} \qquad \exists z.\neg E(z, x_1)$$

postulate that, for a singleton domain, E enumerates all H. Furthermore, together with the above two formulae, the formula

$$\exists z.\ (x_1 \neq x_2) \Rightarrow (E(z, x_1) \Leftrightarrow E(z_1, x_1) \wedge E(z, x_2) \Leftrightarrow E(z_2, x_2))$$

postulates that E enumerates all H over any finite domain. Unfortunately, with this approach, we can never define E to enumerate all H over an infinite domain. Hence second order predicate calculus cannot be reduced to first order one in this way in general.

However, by the finite variability of program variables, given an interval, any program variables V can be generated by finite combination of subintervals of the given one, over each of which V is constantly. Hence, it is possible to construct a 1-ary flexible function, $g(y)$, to enumerate all program variables by the postulates including

$$\lceil\!\rceil \vee \exists y.\lceil g(y) = c\rceil \text{ for any constant } c, \quad \text{and}$$
$$\lceil\!\rceil \vee \exists y.\ \lceil g(y) \Leftrightarrow g(y_1)\rceil; \lceil g(y) \Leftrightarrow g(y_2)\rceil$$

In this way, $\exists V.\phi$ can be reduced to $\exists y_V.dc2il(\phi)$ where $dc2il$ is a translating function from HDC to IL_2 defined later. A complete proof system for HDC can be established based on the completeness result of IL_2 . This idea is hinted by [9].

In order to prove completeness of HDC, we will establish IL_2 , a first-order two-sorted interval temporal logic firstly, in which global variables and functions are divided into two sorts. The rôle of the global variables and rigid functions of the first sort is as usual. The global variables and functions of the second sort and flexible functions of the first sort are used to enumerate program variables and the durations of state expressions in HDC respectively so that we can encode HDC into IL_2 by $dc2il$. Of course, it is not substantial to divide global variables and functions into two sorts, because we can encode many-sorted logic into one-sorted logic by introducing some specific predicates into one-sorted logic to distinguish different objects in the same universe (See [6]). Completeness of IL_2 can be proved with the method used in [5,8]. Because we can show that the consistency of a set of formulae Γ in HDC w.r.t. the proof system of HDC implies the consistency of $dc2il(\Gamma) \cup dc2il(Axiom_{hdc})$ w.r.t. IL_2 , where $Axiom_{hdc}$ stands for the set of all axiom instances for HDC, we can get a model $< \mathcal{F}, \mathcal{J} >$ which satisfies $dc2il(\Gamma) \cup dc2il(Axiom_{hdc})$ by completeness of IL_2 . According to the model $< \mathcal{F}, \mathcal{J} >$, we can construct a model $< \mathcal{F}', \mathcal{I} >$ for HDC which satisfies Γ. Thus, completeness of HDC can be proved.

We will omit the proofs for some lemmas and theorems later in order to save space, but their proofs can be found in [12].

2 Two-Sorted Interval Temporal Logic

In order to prove completeness of HDC on abstract domains, we shall establish IL_2 and then prove its completeness on abstract domains using the method provided in [5,8] in this section.

2.1 Syntax of IL_2

The alphabet of IL_2 includes:

- An infinite set of temporal variables $TVar = \{v_i \mid i \geq 0\}$.
- An infinite set of first sort global variables $Var^1 = \{x_i \mid i \geq 0\}$.
- An infinite set of second sort global variables $Var^2 = \{y_i \mid i \geq 0\}$.
- A special symbol ℓ, which stands for the length of an interval.
- An infinite set of propositional letters $PLetter = \{X_i \mid i \geq 0\}$.
- An infinite set of first sort function symbols $FSymb^1 = \{f_i^n, h_i^n \mid i, n \geq 0\}$. The distinction between f_i^n and h_j^m is that the former is rigid but the latter is flexible.
- A set of second sort flexible function symbols $FSymb^2 = \{g_i^n \mid i, n, \geq 0\}$.
- An infinite set of predicate symbols $RSymb = \{R_i^n \mid i, n \geq 0\}$.
- The connectives $\vee$ and $\neg$.
- The quantifier $\exists$ and the modality ;.

The *terms* of the first sort in IL_2 are defined by the following abstract syntax:

$$\theta \ ::= x \ \mid \ \ell \ \mid \ v \ \mid \ f_i^n(\theta_1, \ldots, \theta_n) \ \mid \ h_i^n(\theta_1, \ldots, \theta_n) \ \mid \ g_i^n(\vartheta_1, \ldots, \vartheta_n)$$

where ϑ_i is a term of the second sort defined as follows:

$$\vartheta \ ::= d \ \mid \ y$$

The *formulae* of IL_2 are defined inductively as follows:

$$\phi \ ::= \ X \ \mid \ R(\theta_1, \ldots, \theta_n) \ \mid \ \neg\phi \ \mid \ \phi \vee \psi \ \mid \ (\phi; \psi) \ \mid \ \exists z.\phi$$

where z stands for any global variable from $Var^1 \cup Var^2$.

A term (formula) is called *rigid* if neither temporal variable, nor ℓ, nor flexible function symbol occurs in it; otherwise called $flexible$. A formula is called *chop free*, if no ";" occurs in it.

2.2 Semantics of IL_2 on Abstract Domains

In this section, we give the meaning of the terms and formulae of IL_2 on abstract domains.

Definition 1. *A* time domain *is a linearly ordered set* $< T, \leq >$.

Definition 2. *Given a time domain* $< T, \leq >$, *we can define a set of intervals* $\mathbf{I}ntv(T) = \{[t_1, t_2] \mid t_1, t_2 \in T \text{ and } t_1 \leq t_2\}$, *where* $[t_1, t_2] = \{t \mid t \in T \text{ and } t_1 \leq t \leq t_2\}$.

Definition 3. *A* duration domain *is a system of the type* $< D, +, 0 >$, *which satisfies the following axioms:*

(D1) $a + (b + c) = (a + b) + c$
(D2) $a + 0 = a = 0 + a$
(D3) $a + b = a + c \Rightarrow b = c, \quad a + c = b + c \Rightarrow a = b$
(D4) $a + b = 0 \Rightarrow a = 0 = b$
(D5) $\exists c.a + c = b \vee b + c = a, \exists c.c + a = b \vee c + b = a$

That is, $< D, +, 0 >$ *is a totally ordered commutative group.*

Definition 4. *Given a time domain $<T, \leq>$ and a duration domain $<D, +, 0>$, a measure m is a function from T to D which satisfies the following conditions:*

(M1) $m([t_1, t_2]) = m([t_1, t_2']) \Rightarrow t_2 = t_2'$
(M2) $m([t_1, t]) + m([t, t_2]) = m([t_1, t_2])$
(M3) $m([t_1, t_2]) = a + b \Rightarrow \exists t.m([t_1, t]) = a \wedge (t_1 \leq t \leq t_2)$

Definition 5. *A frame of* IL_2 *is a quadruple of* $<< T, \leq>, < D, +, 0 >, D_1, m >$, *where* $< T, \leq>$ *is a time domain,* $< D, +, 0 >$ *is a duration domain,* D_1 *is called inhabited domain,* m *is a measure.*

Definition 6. *A model of* IL_2 *is a quintuple with type* $<< T, \leq>, < D, +, 0 >, D_1, m, \mathcal{J} >$, *where* $<< T, \leq>, < D, +, 0 >, D_1, m >$ *is a frame, and* $\mathcal{J}$ *is an* interpretation *of the symbols in* IL_2 *which satisfies the following conditions:* $\mathcal{J}(X) \in \mathbf{I}ntv(T) \to \{0, 1\}$ *for every* $X \in PLetter$; $\mathcal{J}(v) \in \mathbf{I}ntv(T) \to D$ *for every* $v \in TVar$; $\mathcal{J}(R_i^n) \in D^n \to \{0, 1\}$ *for every* $R_i^n \in RSymb$; $\mathcal{J}(f_i^n) \in D^n \to D$ *for every* $f_i^n \in FSymb^1$; $\mathcal{J}(h_i^n) \in D^n \times \mathbf{I}ntv(T) \to D$ *for every* $h_i^n \in FSymb^1$; $\mathcal{J}(g_i^n) \in D_1^n \times \mathbf{I}ntv(T) \to D$ *for every* $g_i^n \in FSymb^2$; *and* $\mathcal{J}(0) = 0, \mathcal{J}(+) = +, \mathcal{J}(=)$ *is* $=$, *and* $\mathcal{J}(\ell) = m$.

Definition 7. *Let* $\mathcal{J}$ *and* $\mathcal{J}'$ *be two interpretations defined as the above.* $\mathcal{J}$ *is* z*-equivalent to* $\mathcal{J}'$ *if* $\mathcal{J}$ *and* $\mathcal{J}'$ *have same values to all symbols, but possibly* z.

Given a model of IL_2 , $<< T, \leq>, < D, +, 0 >, D_1, m, \mathcal{J} >$, and an interval $[t_1, t_2] \in \mathbf{I}\mathrm{ntv}(T)$, the value of a term ϑ or θ can be defined as follows:

$$
\begin{array}{ll}
\mathcal{J}_{t_1}^{t_2}(y) = \mathcal{J}(y) & \text{for } y \in Var^2 \\
\mathcal{J}_{t_1}^{t_2}(x) = \mathcal{J}(x) & \text{for } x \in Var^1 \\
\mathcal{J}_{t_1}^{t_2}(v) = \mathcal{J}(v)([t_1, t_2]) & \text{for } v \in TVar \\
\mathcal{J}_{t_1}^{t_2}(f_i^n(\theta_1, \ldots, \theta_n) = \mathcal{J}(f_i^n)(\mathcal{J}_{t_1}^{t_2}(\theta) \ldots \mathcal{J}_{t_1}^{t_2}(\theta_n)) & \text{for } f_i^n \in FSymb^1 \\
\mathcal{J}_{t_1}^{t_2}(h_i^n(\theta_1, \ldots, \theta_n) = \mathcal{J}(h_i^n)([t_1, t_2], \mathcal{J}_{t_1}^{t_2}(\theta) \ldots \mathcal{J}_{t_1}^{t_2}(\theta_n)) & \text{for } h_i^n \in FSymb^1 \\
\mathcal{J}_{t_1}^{t_2}(g_i^n(\vartheta_1, \ldots, \vartheta_n) = \mathcal{J}(g_i^n)([t_1, t_2], \mathcal{J}_{t_1}^{t_2}(\vartheta) \ldots \mathcal{J}_{t_1}^{t_2}(\vartheta_n)) & \text{for } g_i^n \in FSymb^2
\end{array}
$$

Given a model $\mathcal{M} =< \mathcal{F}, \mathcal{J} >$ where $\mathcal{F} =<< T, \leq>, < D, +, 0 >, D_1, m >$, and an interval $[t_1, t_2]$, the meaning of a formula ϕ is explained by the following rules:

1. $< \mathcal{F}, \mathcal{J} >, [t_1, t_2] \models_{il2} X$ iff $\mathcal{J}(X)([t_1, t_2]) = tt$
2. $< \mathcal{F}, \mathcal{J} >, [t_1, t_2] \models_{il2} R^n(\theta_1, \ldots, \theta_n)$ iff $\mathcal{J}(R^n)(\mathcal{J}_{t_1}^{t_2}(\theta_1), \ldots, \mathcal{J}_{t_1}^{t_2}(\theta_n)) = tt$
3. $< \mathcal{F}, \mathcal{J} >, [t_1, t_2] \models_{il2} \neg\phi$ iff $< \mathcal{F}, \mathcal{J} >, [t_1, t_2] \not\models_{il2} \phi$
4. $< \mathcal{F}, \mathcal{J} >, [t_1, t_2] \models_{il2} \phi \vee \psi$
   iff $< \mathcal{F}, \mathcal{J} >, [t_1, t_2] \models_{il2} \phi$ or $< \mathcal{F}, \mathcal{J} >, [t_1, t_2] \models_{il2} \psi$
5. $< \mathcal{F}, \mathcal{J} >, [t_1, t_2] \models_{il2} \phi; \psi$
   iff $< \mathcal{F}, \mathcal{J} >, [t_1, t] \models_{il2} \phi$ and $< \mathcal{F}, \mathcal{J} >, [t, t_2] \models_{il2} \psi$ for some $t \in [t_1, t_2]$
6. $< \mathcal{F}, \mathcal{J} >, [t_1, t_2] \models_{il2} \exists z.\phi$ iff $< \mathcal{F}, \mathcal{J}' >, [t_1, t_2] \models_{il2} \phi$ for some interpretation $\mathcal{J}'$ which is z-equivalent to $\mathcal{J}$

Satisfaction and validity can be defined in the usual way, see [12].
The following abbreviations will be used:

$\Diamond\phi \ \widehat{=}\ \mathbf{true};(\phi;\mathbf{true})$ reads: "for some sub-interval: ϕ"
$\Box\phi \ \widehat{=}\ \neg\Diamond(\neg\phi)$ reads: "for all sub-intervals: ϕ"

Furthermore, the standard abbreviations from predicate logic will be used. When $\neg, \exists z, \Box,$ and $\Diamond$ occur in formulae they have higher precedence than the binary connectives and the modality ;. The modality ; has higher precedence than the binary connectives.

Definition 8. *Let Φ be an* IL_2 *formula, and let $\mathcal{M} =< \mathcal{F}, \mathcal{J} >$ be a model of* IL_2 *, where $\mathcal{F} =<< T, \leq>, < D, +, 0 >, D_1, m >$ is the corresponding frame. Φ is said to have the* finite variability *on $\mathcal{M}$ if for every $[t_1, t_2] \in \mathbf{I}ntv(T)$, $\mathcal{M}, [t_1, t_2] \models_{il2} \Phi \Rightarrow \Box\Phi$ and there exist $t'_1, \ldots, t'_n$, such that $t_1 = t'_1 \leq \ldots \leq t'_n = t_2$ and for all $i = 1, \ldots, n-1$ $\mathcal{M}, [t'_i, t'_{i+1} \models_{il2} \Box\Phi$. Φ is said to have finite variability on a class of models $\mathcal{K}$ if it has the property on every member of $\mathcal{K}$.*

Definition 9. *Let Φ be an* IL_2 *formula. We define the sequence of formulae $\{\Phi^k\}_{k<\omega}$ as follows:*

$$\Phi^0 \ \widehat{=}\ \ell = 0, \qquad \Phi^{k+1} \ \widehat{=}\ (\Phi^k;\Box\Phi)$$

For the rest of this section we will fix a set of IL_2 formulae Ω and consider only IL_2 models on which Φ has finite variability for every $\Phi \in \Omega$. We will use $\mathcal{K}_\Omega$ to denote the class of models that satisfy the above property later. So the following proof system takes Ω as a parameter. Of course, all discussions below can be applied to an arbitrary set of IL_2 formulae Ω. If $\Omega = \emptyset$, then the case is same as in [5]. The finite variability of Φ means that for any interval one can partition the interval into finitely many subintervals such that $\Box\Phi$ holds for each of the subintervals. The axiom ITL_Ω and rule IR^Φ given below are used to axiomatise the finite variability of all $\Phi \in \Omega$.

2.3 Proof System of IL_2 with Ω

In this section, we give a sound and complete proof system of IL_2 with Ω w.r.t. $\mathcal{K}_\Omega$. The notation $\vdash_{il2_\Omega} \phi$ means that ϕ is provable, i.e. that ϕ is a theorem of IL_2 with Ω.

Definition 10. *A term θ is called* free for x *in ϕ if x does not occur freely in ϕ within a scope of $\exists x'$ or $\forall x'$ where x' is any variable occurring in θ.*

The axioms of IL_2 are:

ITL1: $\ell \geq 0$
ITL2: $((\phi;\psi) \wedge \neg(\phi;\varphi)) \Rightarrow (\phi;(\psi \wedge \neg\varphi))$
$((\phi;\psi) \wedge \neg(\varphi;\psi)) \Rightarrow ((\phi \wedge \neg\varphi);\psi)$
ITL3: $((\phi;\psi);\varphi) \Leftrightarrow (\phi;(\psi;\varphi))$
ITL4: $(\phi;\psi) \Rightarrow \phi$ if ϕ is a rigid formula
$(\phi;\psi) \Rightarrow \psi$ if ψ is a rigid formula

$$
\begin{array}{ll}
\text{ITL5:} & (\exists z.\phi;\psi) \Rightarrow \exists z.(\phi;\psi) \quad \text{if } z \text{ is not free in } \psi \\
& (\phi;\exists z.\psi) \Rightarrow \exists z.(\phi;\psi) \quad \text{if } z \text{ is not free in } \phi \\
\text{ITL6:} & ((\ell = a);\phi) \Rightarrow \neg((\ell = a);\neg\phi) \\
& (\phi;(\ell = a)) \Rightarrow \neg(\neg\phi;(\ell = a) \\
\text{ITL7:} & (a \geq 0 \wedge b \geq 0) \Rightarrow ((\ell = a + b) \Leftrightarrow ((\ell = a);(\ell = b))) \\
\text{ITL8:} & \phi \Rightarrow (\phi;(\ell = 0)) \\
& \phi \Rightarrow ((\ell = 0);\phi) \\
\text{ITL}_\Omega : & \Phi \Rightarrow \Box\Phi \qquad \text{for all } \Phi \in \Omega
\end{array}
$$

The inference rules of IL_2 are:

$$
\begin{array}{ll}
\text{N:} & \begin{array}{l}\text{if } \phi \text{ then } \neg(\neg\phi;\psi) \\ \text{if } \psi \text{ then } \neg(\psi;\neg\phi)\end{array} \qquad
\text{M:} \quad \begin{array}{l}\text{if } \phi \Rightarrow \psi \text{ then } (\phi;\varphi) \Rightarrow (\psi;\varphi) \\ \text{if } \phi \Rightarrow \psi \text{ then } (\varphi;\phi) \Rightarrow (\varphi;\psi)\end{array} \\
\text{IR}^{\Phi} & \dfrac{H(\Phi^0/X) \quad \forall k{<}\omega.H(\Phi^k/X) \Rightarrow H(\Phi^{k+1}/X)}{H(\mathbf{true}/X)} \qquad \text{for } \Phi \in \Omega
\end{array}
$$

The proof system of IL_2 with Ω also contains all axioms and rules for propositional logic, predicate logic, and real arithmetic, such as

$$(\text{G}): \text{ if } \phi \text{ then } \forall z.\phi$$

However, for the following axiom, side condition is necessary.

$$(\text{Q}) : \forall x.\phi(x) \Rightarrow \phi(\theta) \begin{cases} \text{if either } \theta \text{ is free for } x \text{ in } \phi(x) \text{ and } \theta \text{ is rigid} \\ \text{or } \theta \text{ is free for } x \text{ in } \phi(x) \text{ and } \phi(x) \text{ is chop free.} \end{cases}$$

An explanation of the necessity of the side condition is given in [11]. The axioms and rules for equality, addition, etc. in real arithmetic will not be listed here, but can be found in [11].

Theorem 1 (Soundness). *The proof system is sound, i.e.* $\vdash_{il2_\Omega} \phi$ *implies* $\models_{il2_\Omega} \phi$*, where* $\models_{il2_\Omega} \phi$ *means* ϕ *is valid on every model* $\mathcal{M} \in \mathcal{K}_\Omega$*.*

Definition 11. *Given a set of* IL_2 *formulae* Γ*, if* $\Gamma \nvdash_{il2_\Omega}$ **false***, then* Γ *is called* consistent *w.r.t. the proof system of* IL_2 *with* Ω*, otherwise called* inconsistent*.*

Theorem 2 (Completeness). *Given a set of* IL_2 *formulae* Γ*. if* Γ *is consistent w.r.t. the proof system of* IL_2 *with* Ω*, then there exists a model* $\mathcal{M} = << T, \leq >, < D, +, 0 >, D_1, m, \mathcal{J} >$ *on which* Φ *has finite variability for every* $\Phi \in \Omega$*, and an interval* $[t_1, t_2] \in \mathbf{I}ntv(T)$ *such that* $\mathcal{M}, [t_1, t_2] \models_{il2_\Omega} \Gamma$*.*

Proof. Using the method provided in [5,8], it can be proved. See [12]. □

3 Higher-Order Duration Calculus

In this section, we establish a higher-order duration calculus, which is an extension of the original duration calculus, by introducing program variables and quantifications over them.

3.1 Syntax of HDC

The alphabet of HDC contains all symbols of IL_2 except for the symbols of the second sort and the flexible function symbols of the first sort. Besides, it also includes an infinite set of program variables $PVar = \{V_i \mid i \geq 0\}$. In HDC, all temporal variables have a special structure $\int S$ where S is a state expression defined as follows:

$$S \quad ::= \quad 0 \quad | \quad 1 \quad | \quad S_1 \vee S_2 \quad | \quad \neg S \quad | \quad R(\vartheta_1, \ldots, \vartheta_n)$$

where R is the characteristic function of predicate R, and $\vartheta_1, \ldots, \vartheta_n$ are called state terms defined as:

$$\vartheta ::= x \quad | \quad V \quad | \quad f(\vartheta_1, \ldots, \vartheta_n)$$

The *terms* of HDC are constructed as follows:

$$\theta \quad ::= x \quad | \quad \ell \quad | \quad \overleftarrow{\vartheta} \quad | \quad \overrightarrow{\vartheta} \quad | \quad v \quad | \quad f(\theta_1, \ldots, \theta_n)$$

where v has the form $\int S$ where S is a state expression defined above, and "$\leftarrow$" and "$\rightarrow$" are two special functions with a domain of state terms and a codomain of functions from the intervals to duration domain.

The *formulae* of HDC are defined inductively as follows:

$$\phi \quad ::= \quad X \quad | \quad R(\theta_1, \ldots, \theta_n) \quad | \quad \neg\phi \quad | \quad \phi \vee \psi \quad | \quad (\phi; \psi) \quad | \quad \exists x.\phi \quad | \quad \exists V.\phi$$

A state term (term or formula) is called *rigid* if neither program variable nor ℓ occurs in it; otherwise called *flexible*.

Remark 1. We can show that a rigid state expression is also a rigid formula by the above definitions if we do not distinguish predicate and its characteristic function. For example, $(x + 3 > 1)$ can be taken as a state as well as a formula according to the syntactic definitions above. In order to avoid confusion, when S is rigid, we will use ϕ_S to stand for the rigid formula corresponding to S.

3.2 Semantics of HDC

In this subsection, we give the meaning of terms and formulae in HDC on abstract domains. HDC frames are essentially IL_2 frames too, but a slight difference is that there is no inhabited domain in HDC frames.

Definition 12. *A model of HDC is a quadruple with type* $<<T, \leq>, <D, +, 0>, m, \mathcal{I}>$, *where* $<<T, \leq>, <D, +, 0>, m>$ *is a frame, and* $\mathcal{I}$ *is an* interpretation *of the symbols in HDC which satisfies the following condition:*

For every $V \in PVar$, *and every* $[t_1, t_2] \in \mathbf{I}ntv(T)$ *there exists* $t'_1, \ldots, t'_n$ *such that* $t_1 = t'_1 \leq \ldots \leq t'_n = t_2$, *and for any* $t, t' \in [t'_i, t'_{i+1})$ *implies* $\mathcal{I}(V)(t) = \mathcal{I}(V)(t')$.

This property is known as the finite variability of program variables.

Given a model of HDC $\mathcal{M} =<< T, \leq>, < D, +, 0 >, m, \mathcal{I} >$, the meaning of program variables and propositional letters is given as: $\mathcal{I}(V)T \to D$ and $\mathcal{I}(X) \in \mathbf{I}\text{ntv}(T) \to \{tt, ff\}$ respectively.

The semantics of a state term ϑ, given a model $\mathcal{M} =< \mathcal{F}, \mathcal{I} >$, is a function with type $T \to D$ defined inductively on its structure as follows:

$$\begin{array}{ll} \mathcal{I}(x)(t) & = \mathcal{I}(x) \\ \mathcal{I}(V)(t) & = \mathcal{I}(V)(t) \\ \mathcal{I}(f^n(\vartheta_1, \ldots, \vartheta_n))(t) & = \mathcal{I}(f^n)(\mathcal{I}(\vartheta_n)(t), \ldots, \mathcal{I}(\vartheta_n)(t)) \end{array}$$

The semantics of a state expression S, given a model $\mathcal{M} =< \mathcal{F}, \mathcal{I} >$, is a function with type $T \to \{0, 1\}$ defined inductively on its structure as follows:

$$\begin{array}{ll} \mathcal{I}(0)(t) & = 0 \\ \mathcal{I}(1)(t) & = 1 \\ \mathcal{I}(R^n(\vartheta_1, \ldots, \vartheta_n))(t) & = \mathcal{I}(R^n)(\mathcal{I}(\vartheta_1)(t), \ldots, \mathcal{I}(\vartheta_n)(t)) \\ \mathcal{I}(\neg S)(t) & = 1 - \mathcal{I}(S)(t) \\ \mathcal{I}(S_1 \vee S_2)(t) & = \begin{cases} 0 \text{ if } \mathcal{I}(S_1)(t) = 0 \text{ and } \mathcal{I}(S_2)(t) = 0 \\ 1 \text{ otherwise} \end{cases} \end{array}$$

Lemma 1. *Let S be a state expression and $\mathcal{I}$ be an interpretation of the symbols in HDC on a frame $\mathcal{F} =<< T, \leq>, < D, +, 0 >, m >$. Then for every $[t_1, t_2] \in \mathbf{I}ntv(T)$ there exist $t'_1, \ldots, t'_n$ such that $t_1 = t'_1 \leq \ldots \leq t'_n = t_2$, and for any $t, t' \in [t'_i, t'_{i+1})$ implies $\mathcal{I}(S)(t) = \mathcal{I}(S)(t')$ for all $i = 1, \ldots, n-1$.*

Proof. Induction on the construction of S. □

Using Lemma 1, we can give the interpretation of $\int S$ under an HDC model $\mathcal{M} =< \mathcal{F}, \mathcal{I} >$. Let $[t_1, t_2] \in \mathbf{I}\text{ntv}(T)$ and $t'_1, \ldots, t'_n$ be a partition of $[t_1, t_2]$ which have the property stated in Lemma 1. We define $p \bullet c$ for $p \in \{0, 1\}$ and $c \in D$ as follows:

$$p \bullet c \quad = \quad \begin{cases} 0 & \text{if } p = 0 \\ c & \text{if } p = 1 \end{cases}$$

Then $\mathcal{I}(\int S)([t_1, t_2]) = \sum_{i=1}^{n-1} \mathcal{I}(S)(t'_i) \bullet m([t'_i, t'_{i+1}])$. It is easy to show that this definition does not depend on the particular choice $t'_1, \ldots, t'_n$.

Given a model $\mathcal{M} =< \mathcal{F}, \mathcal{I} >$, and an interval $[t_1, t_2] \in \mathbf{I}\text{ntv}(T)$, the meaning of initial and final values of state terms $\overleftarrow{\vartheta}, \overrightarrow{\vartheta}, \overleftarrow{\vartheta_1}, \overrightarrow{\vartheta_1}, \ldots$, are functions with type $\mathbf{I}\text{ntv}(T) \to D$ defined as follows:

$$\mathcal{I}(\overleftarrow{\vartheta}, [t_1, t_2]) = d, \textit{ iff } < \mathcal{F}, \mathcal{I} >, [t_1 - \delta, t_1] \vdash_{hdc} \lceil \vartheta = d \rceil, \text{ for some } \delta > 0.$$
$$\mathcal{I}(\overrightarrow{\vartheta}, [t_1, t_2]) = d, \textit{ iff } < \mathcal{F}, \mathcal{I} >, [t_2, t_2 + \delta] \vdash_{hdc} \lceil \vartheta = d \rceil, \text{ for some } \delta > 0.$$

where $\lceil S \rceil \ \widehat{=} \ \int S = \ell \wedge \ell > 0$. It means that S takes value 1 almost everywhere in a non-point interval. We will use $\lceil \rceil$ to stand for $\ell = 0$.

The meaning of other syntactic entities in HDC can be given similarly to the ones in IL_2 , and other notions for HDC also can be defined similarly.

3.3 Proof System of HDC

In this section, we give a proof system of HDC. The notation $\vdash_{hdc} \phi$ means that ϕ is provable.

The proof system of HDC includes all axioms and inference rules in IL_2 but the axiom ITL_Ω. Besides, it also includes the following three groups of axioms and rules.

The first group is used to specify how to calculate and reason about state durations. They are:

$$\begin{array}{llll}
\text{(DC1)} & \int 0 = 0 & \text{(DC4)} & \int S_1 + \int S_2 = \int(S_1 \vee S_2) + \int(S_1 \wedge S_2) \\
\text{(DC2)} & \int 1 = \ell & \text{(DC5)} & ((\int S = x_1)^\frown(\int S = x_2)) \Rightarrow (\int S = x_1 + x_2) \\
\text{(DC3)} & \int S \geq 0 & \text{(DC6)} & \int S_1 = \int S_2, \text{ if } S_1 \Leftrightarrow S_2 \\
 & & \text{(DC7)} & \lceil S \rceil \ \Leftrightarrow \ (\phi_S \wedge \ell > 0), \text{ if } S \text{ is rigid}
\end{array}$$

The rôle of the second group is to calculate the initial and final values of ϑ, $\overleftarrow{\vartheta}$ and $\overrightarrow{\vartheta}$. They are:

$$\begin{array}{ll}
\text{(PV1)} & (\ell > 0); ((\overleftarrow{\vartheta} = x_1) \wedge (\ell = x_2)) \ \Leftrightarrow \ \mathbf{true}; \lceil \vartheta = x_1 \rceil; (\ell = x_2) \\
\text{(PV2)} & ((\overrightarrow{\vartheta} = x_1) \wedge (\ell = x_2)); (\ell > 0) \ \Leftrightarrow \ (\ell = x_2); \lceil \vartheta = x_1 \rceil; \mathbf{true}
\end{array}$$

PV1 and PV2 formulate the meaning of the initial value and final value of a state term which are inherited from the previous statement, and passed to the next one. Because the function $\leftarrow$ ($\rightarrow$) involves the value of a state term at left neighbourhood (right neighbourhood), the neighbourhood rule is necessary in order to axiomatise them.

$$\text{NR} \quad \text{If } \ (\ell = a); \Psi; (\ell = b) \Rightarrow (\ell = a); \Upsilon; (\ell = b), \quad \text{then } \Psi \Rightarrow \Upsilon. \ (a, b \geq 0)$$

Remark 2. This rule can be looked as a rule of IL_2 . Although the rule will destroy the deduction theorem of IL_2 , IL_2 will keep completeness after introducing it.

The last group is used to specify the semantics of V, $\overleftarrow{V}$ and $\overrightarrow{V}$ in the context of quantifications.

The axiom and rule below are standard as in predicate logic.

$$\begin{array}{ll}
\mathrm{G_V}: & \text{if } \phi \text{ then } \forall V.\phi \\
\mathrm{Q_V}: & \forall V.\phi(V) \ \Rightarrow \ \phi(\vartheta)
\end{array}$$

If $\overleftarrow{V}$ ($\overrightarrow{V}$) does not occur in formula ϕ, then it can take any value, since the value of $\overleftarrow{V}$ ($\overrightarrow{V}$) is defined by value of V outside the reference interval w.r.t. ϕ. Hence,

$$\begin{array}{lll}
\text{(HDC1)} \ \exists V.\phi \ \Rightarrow \ \exists V. \ \phi \wedge (\overleftarrow{V} = x) & & \text{if } \overleftarrow{V} \notin \phi \\
\text{(HDC2)} \ \exists V.\phi \ \Rightarrow \ \exists V. \ \phi \wedge (\overrightarrow{V} = x) & & \text{if } \overrightarrow{V} \notin \phi
\end{array}$$

The distributivity of $\exists V$ over the chop operator is the most essential property of V as a function over time. $\exists V$ can distribute over the chop, if and only if the value of $\overrightarrow{V}$ in the left operand of the chop can match the value of V in the right operand, and symmetrically for the value of $\overleftarrow{V}$ in the right operand. That is,

$$\text{(HDC3)} \quad \left(\begin{array}{l} (\exists V.\ \phi \wedge (\mathit{true}; \lceil V = x_1 \rceil \vee \lceil \rceil) \wedge (\overrightarrow{V} = x_2)) \\ ; (\exists V.\ \psi \wedge (\lceil V = x_2 \rceil; \mathbf{true} \vee \lceil \rceil) \wedge (\overleftarrow{V} = x_1)) \end{array} \right) \ \Rightarrow\ \exists V.\ \phi; \psi$$

When $\overrightarrow{V} \notin \phi$ or $\overleftarrow{V} \notin \psi$, it can be derived from the above axioms that $(\exists V.\phi); \exists V.\psi \ \Rightarrow\ \exists V.\ \phi; \psi$

In order to define program variables as finitely varied functions, in the proof system, we let $\Omega = \Omega_{hdc}$ where $\Omega_{hdc} \ \widehat{=}\ \{\exists x(\lceil V = x \rceil \vee \lceil\ \rceil) \mid V \in PVar\}$.

Theorem 3 (Soundness). *The proof system of HDC is sound, i.e.* $\vdash_{hdc} \phi$ *implies* $\models_{hdc} \phi$

4 Completeness of HDC on Abstract Domains

In this section, we will apply completeness of IL_2 with Ω to show HDC is complete on abstract domains. To this end, let us choose a language $\mathcal{L}_{IL_2}$ for IL_2 with four special flexible function symbols $\oplus$, $\ominus$, h_r and h_l, in which there is only one unary function symbol g of the second sort, and a language $\mathcal{L}_{hdc}$ for HDC. $\oplus$ and $\ominus$ have type $((\mathbf{I}\mathrm{ntv}(T) \to D) \times (\mathbf{I}\mathrm{ntv}(T) \to D)) \to (\mathbf{I}\mathrm{ntv}(T) \to D)$, and h_l and h_r have type $(\mathbf{I}\mathrm{ntv}(T) \to D) \to (\mathbf{I}\mathrm{ntv}(T) \to D)$. In $\mathcal{L}_{IL_2}$, the definition of terms will be extended by allowing that duration terms and neighbourhood terms are also terms, where duration terms are defined as: $h(\theta_1, \ldots, \theta_n)$ is duration term; If t_1 and t_2 are duration terms then $t_1 \oplus t_2$ and $t_1 \ominus t_2$ are both duration terms too, neighbourhood terms are defined as: If nt is of the forms x or $g(y)$ or $f(nt_1, \ldots, nt_n)$ then $h_r(nt)$ and $h_l(nt)$ are both neighbourhood terms. It is easy to define the meaning of the above extensions using the usual way. Obviously, IL_2 with Ω is still complete after extending. We will use duration terms to correspond the terms of state durations, neighbourhood terms to correspond the left and right values of state terms in the below translation $dc2il$ from $\mathcal{L}_{hdc}$ to $\mathcal{L}_{IL_2}$.

Let us fix two bijections: $V \to y_V$, and $R \to h_R$ between $\mathcal{L}_{hdc}$ and $\mathcal{L}_{IL_2}$. We will establish a bijection between $\mathcal{L}_{hdc}$ and a subset of $\mathcal{L}_{IL_2}$ by function $dc2il$ from $\mathcal{L}_{hdc}$ to a subset of $\mathcal{L}_{IL_2}$ and its inverse $il2dc$.

We can prove that if a set of formulae Γ in $\mathcal{L}_{hdc}$ is consistent w.r.t. the proof system of HDC then $dc2il(\Gamma)$ plus $dc2il(Axiom_{hdc})$ where $Axiom_{hdc}$ contains all axiom instances of HDC is consistent w.r.t. the proof system of IL_2 with $dc2il(\Omega_{hdc})$. From now on, let $\Omega = dc2il(\Omega_{hdc})$. By Theorem 2, we can get a model $< \mathcal{F}, \mathcal{J} >$ and an interval $[t_1, t_2]$ such that $< \mathcal{F}, \mathcal{J} >, [t_1, t_2] \models_{il2_\Omega} dc2il(\Gamma) \cup dc2il(Axiom_{hdc})$. Finally, according to the model and interval, we can construct a model $< \mathcal{F}', \mathcal{I} >$ for HDC such that $< \mathcal{F}, \mathcal{I} >, [t_1, t_2] \models_{hdc} \Gamma$.

We define the translating function $dc2il$ from $\mathcal{L}_{hdc}$ to $\mathcal{L}_{IL_2}$ as follows:

$$dc2il(\vartheta) \;\widehat{=}\; \begin{cases} x & \text{if } \vartheta = x \\ g(y_V) & \text{if } \vartheta = V \\ f(dc2il(\vartheta_1), \ldots, dc2il(\vartheta_n)) & \text{if } \vartheta = f(\vartheta_1, \ldots, \vartheta_n) \end{cases}$$

$$dc2il(\theta) \;\widehat{=}\; \begin{cases} x & \text{if } \theta = x \\ h_l(dc2il(\vartheta)) & \text{if } \theta = \overleftarrow{\vartheta} \\ h_r(dc2il(\vartheta)) & \text{if } \theta = \overrightarrow{\vartheta} \\ f(dc2il(\theta_1), \ldots, dc2il(\theta_n)) & \text{if } \theta = f(\theta_1, \ldots, \theta_n) \end{cases}$$

$$dc2il(\textstyle\int S) \;\widehat{=}\; \begin{cases} h_0 & \text{if } S = 0 \\ h_1 & \text{if } S = 1 \\ h_R(dc2il(\vartheta_1), \ldots, dc2il(\vartheta_n)) & \text{if } S = R(\vartheta_1, \ldots, \vartheta_n) \\ \ell - dc2il(\int S_1) & \text{if } S = \neg S_1 \\ dc2il(\int S_1) \ominus dc2il(\int S_2) & \text{if } S = S_1 \wedge S_2 \\ dc2il(\int S_1) \oplus dc2il(\int S_2) & \text{if } S = S_1 \vee S_2 \end{cases}$$

$$dc2il(\phi) \;\widehat{=}\; \begin{cases} X & \text{if } \phi = X \\ R(dc2il(\theta_1), \ldots, dc2il(\theta_n)) & \text{if } \phi = R(\theta_1, \ldots, \theta_n) \\ \neg dc2il(\psi) & \text{if } \phi = \neg\psi \\ dc2il(\phi_1) \vee dc2il(\phi_2) & \text{if } \phi = \phi_1 \vee \phi_2 \\ dc2il(\phi_1) \wedge dc2il(\phi_2) & \text{if } \phi = \phi_1 \wedge \phi_2 \\ \exists x.dc2il(\psi) & \text{if } \phi = \exists x.\psi \\ \exists y_V.dc2il(\psi) & \text{if } \phi = \exists V.\psi \end{cases}$$

where $h_0 = 0$ and $h_1 = \ell$.

Symmetrically, we define its inverse $il2dc$ as follows:

$$il2dc(\theta) \;\widehat{=}\; \begin{cases} x & \text{if } \theta = x \\ V & \text{if } \theta = g(y_V) \\ \overleftarrow{il2dc(\theta)} & \text{if } \theta = h_l(\theta) \\ \overrightarrow{il2dc(\theta)} & \text{if } \theta = h_r(\theta) \\ f(il2dc(\theta_1), \ldots, il2dc(\theta_n)) & \text{if } \theta = f(\theta_1, \ldots, \theta_n) \\ \int 0 & \text{if } \theta = h_0 \\ \int 1 & \text{if } \theta = h_1 \\ \int R(il2dc(\theta_1), \ldots, il2dc(\theta_n)) & \text{if } \theta = h_R(\theta_1, \ldots, \theta_n) \end{cases}$$

$il2dc(h_{R_1}(\theta_{11}, \ldots, \theta_{1n_1}) * \ldots * h_{R_m}(\theta_{m1}, \ldots, \theta_{mn_m})) \;\widehat{=}\; \int(R_1(il2dc(\theta_{11}), \ldots, il2dc(\theta_{1n_1}))\& \ldots \& R_m(il2dc(\theta_{m1}), \ldots, il2dc(\theta_{mn_m})))$ where $* \in \{\oplus, \ominus\}$ and $\& \in \{\vee, \wedge\}$. If $* = \oplus$ then the corresponding $\& = \vee$, otherwise the corresponding $\& = \wedge$.

$$il2dc(\phi) \;\widehat{=}\; \begin{cases} X & \text{if } \phi = X \\ R(il2dc(\theta_1), \ldots, il2dc(\theta_n)) & \text{if } \phi = R(\theta_1, \ldots, \theta_n) \\ \neg il2dc(\psi) & \text{if } \phi = \neg\psi \\ il2dc(\phi_1) \vee il2dc(\phi_2) & \text{if } \phi = \phi_1 \vee \phi_2 \\ il2dc(\phi_1) \wedge il2dc(\phi_2) & \text{if } \phi = \phi_1 \wedge \phi_2 \\ \exists x.il2dc(\psi) & \text{if } \phi = \exists x.\psi \\ \exists V.il2dc(\psi) & \text{if } \phi = \exists y_V.\psi \end{cases}$$

From the definitions of $dc2il$ and $il2dc$ above, we have the following result.

Theorem 4. *For any set of formulae $\Gamma \subset \mathcal{L}_{hdc}$, Γ is consistent w.r.t. the proof system of HDC iff $dc2il(\Gamma) \cup dc2il(Axiom_{hdc})$ is consistent w.r.t. the proof system of* IL_2 *with $dc2il(\Omega_{hdc})$.*

Proof. By the above definitions of $dc2il$ and $il2dc$, it is trivial. □

Theorem 5. *If Γ is consistent w.r.t. the proof system of HDC, then Γ is satisfiable.*

Proof. The consistency of Γ w.r.t. HDC implies the consistency of $\Gamma_0 = \{\ell = a\}; \Gamma; \{\ell = b\}$ w.r.t. HDC where $a, b > 0$ by the neighbourhood rule. The consistency of Γ_0 w.r.t. HDC implies the consistency of $dc2il(\Gamma_0) \cup dc2il(Axiom_{hdc})$ w.r.t. IL_2 with Ω by Theorem 4. Hence, by Theorem 2, there exists an IL_2 model $\mathcal{M} =< \mathcal{F}, \mathcal{J} >$ on which Φ has the finite variability property for every $\Phi \in \Omega$, where $\mathcal{F} =<< T, \leq >, < D, +, 0 >, D_1, m >$ is its frame, and an interval $[t_1, t_2] \in \mathbf{I}\text{ntv}(T)$ such that $< \mathcal{F}, \mathcal{J} >, [t_1, t_2] \models_{il2_\Omega} dc2il(\Gamma_0) \cup dc2il(Axiom_{hdc})$. Hence, there exists a proper sub-interval $[t_1', t_2']$ such that $t_1 < t_1' \leq t_2' < t_2$, $t_1' = t_1 + a$, $t_2' = t_2 - b$, and $< \mathcal{F}, \mathcal{J} >, [t_1', t_2'] \models_{il2_\Omega} dc2il(\Gamma) \cup dc2il(Axiom_{hdc})$.

From now on, we prove that there exists a model $< \mathcal{F}', \mathcal{I} >$ of HDC such that $< \mathcal{F}', \mathcal{I} >, [t_1', t_2'] \models_{hdc} \Gamma$.

Let $\Im$ be a class of interpretations of IL_2 such that for every element $\mathcal{J}' \in \Im$, $< \mathcal{F}, \mathcal{J}' >$ is a model of IL_2 , and $\mathcal{J}'[\![g]\!] = \mathcal{J}[\![g]\!]$

For every $\mathcal{J}' \in \Im$ we construct an interpretation $\mathcal{I}'$ of $\mathcal{L}_{hdc}$ as follows:

For every $V \in PVar$, the formula $\Phi = dc2il(\exists x.\lceil V = x \rceil \vee \lceil\!\rceil) \in dc2il(\Omega_{hdc})$. By Theorem 2, there exists a partition $t_1 = t_1'' \leq t_2'' \leq \ldots \leq t_n'' = t_2$ of $[t_1, t_2]$ such that $< \mathcal{F}, \mathcal{J}' >, [t_i'', t_{i+1}''] \models_{il2_\Omega} dc2il(\exists x.\lceil V = x \rceil \vee \lceil\!\rceil)$, i.e. $< \mathcal{F}, \mathcal{J}' >, [t_i'', t_{i+1}''] \models_{il2_\Omega} \exists x.\lceil h_{id}(g(y_V), x) \rceil \vee \lceil\!\rceil$, for $i = 1, \ldots, n-1$. Thus $\mathcal{I}'$ can be defined as follows:

$$
(\bullet\bullet)\quad
\begin{array}{l}
\mathcal{I}'[\![V]\!](t) \mathrel{\widehat{=}} \begin{cases} \mathcal{J}''(x) & \text{if } t_i'' \leq t < t_{i+1}'', \text{and} \\ & < \mathcal{F}, \mathcal{J}'' >, [t_i'', t_{i+1}''] \models_{il2_\Omega} \lceil h_{id}(g(y_V), x) \rceil \vee \lceil\!\rceil \\ & \text{where } \mathcal{J}'' \text{ is } x\text{-equivalent to } \mathcal{J}' \\ 0 & \text{otherwise} \end{cases} \\
\mathcal{I}'(x) \mathrel{\widehat{=}} \mathcal{J}'(x) \\
\mathcal{I}'(f_i^n) \mathrel{\widehat{=}} \mathcal{J}'(f_i^n) \\
\mathcal{I}'(X) \mathrel{\widehat{=}} \mathcal{J}'(X) \\
\mathcal{I}'(R_i^n) \mathrel{\widehat{=}} \mathcal{J}'(R_i^n)
\end{array}
$$

In order to prove the theorem, we need the following two lemmas.

Lemma 2. *Let $\mathcal{J}' \in \Im$, and $\mathcal{J}'$ and $\mathcal{I}'$ have the relation ($\bullet\bullet$). Then for any term θ in $\mathcal{L}_{hdc}$ and any interval $[c, d] \subseteq [t_1', t_2']$, we have:*

$$\mathcal{I}'[\![\theta]\!][c, d] = \mathcal{J}'[\![dc2il(\theta)]\!][c, d].$$

Proof of the lemma: See [12]. □

Now, we can give a correspondence between $\mathcal{I}'$ and $\mathcal{J}'$ which have the relation ($\bullet\bullet$) on formulae by the following lemma.

Lemma 3. *Let $\mathcal{J}' \in \Im$, and $\mathcal{J}'$ and $\mathcal{I}'$ have the relation (●●). Then for any formula ϕ in $\mathcal{L}_{hdc}$, and any subinterval $[c,d] \subset [t'_1, t'_2]$, $< \mathcal{F}, \mathcal{I}' >, [c,d] \models_{hdc} \phi$ iff $< \mathcal{F}, \mathcal{J}' >, [c,d] \models_{il2_\Omega} dc2il(\phi)$.*

Proof of the lemma: We give its proof by induction on the construction of ϕ. We only prove the case $\phi = \exists V.\psi$, the other cases can be proved easily by Lemma 2 and the definition of *dc2il*.

"$\Leftarrow$" It is easy to show.

"$\Rightarrow$" Let $< \mathcal{F}, \mathcal{I}' >, [c,d] \models_{hdc} \phi$. Then there exists an interpretation $\mathcal{I}''$ for HDC which is V-equivalent to $\mathcal{I}'$, and $< \mathcal{F}, \mathcal{I}'' >, [c,d] \models_{hdc} \psi$. Let $t''_0, t''_1, \ldots, t''_n, t''_{n+1} \in T$ such that $t_1 \leq t''_0 < c = t''_1 \leq \ldots \leq t''_n = d < t''_{n+1} \leq t_2$ and $\mathcal{I}''(V)$ is constant on $[t'_i, t'_{i+1})$ for $i = 0, \ldots, n$, and assume these $n+1$ constants are $c_0, \ldots, c_n$. The above assumption is reasonable because $< \mathcal{F}, \mathcal{I}'' >$ is a model of HDC. Since $\mathcal{M} =< \mathcal{F}, \mathcal{J} >$ is a model of IL_2 , by the axiom $\mathrm{Q_V}$ we have that for all $i = 0, \ldots, n$, there exists some $d_i \in D_1$ such that

$$(*)\ \mathcal{J}[\![\, g \,]\!](d_i, [b,e]) = c_i \text{ if } \mathcal{I}''[\![\, V \,]\!] = c_i$$

for any sub-interval $[b,e] \subseteq [t''_i, t''_{i+1}]$.

Applying the axioms HDC1-HDC3 n times implies that there exists a $d \in D_1$ such that for all $i = 0, \ldots, n+1$ and $t \in [t''_i, t''_{i+1})$

$$\mathcal{I}''(V)(t) = c_i \quad \text{iff} \quad < \mathcal{F}, \mathcal{J}'' >, [t''_i, t''_{i+1} \models_{il2_\Omega} [\![h_{id}(g(d), c_i)]\!]$$

Let $\mathcal{J}''(z) = \mathcal{J}'(z)$ for all symbols in IL_2 but y_V, and $\mathcal{J}''(y_V) = d$ defined by the above. Hence $\mathcal{J}''$ is y_V-equivalent to $\mathcal{J}'$, and $\mathcal{J}''$ and $\mathcal{I}''$ have the definition relation given in (●●). By the induction hypothesis, $\mathcal{J}'', [c,d] \models_{il2_\Omega} dc2il(\psi)$, whence $\mathcal{J}', [c,d] \models_{il2_\Omega} dc2il(\phi)$ by the definition of *dc2il*. □

Now, let $\mathcal{F}' =<< T, \leq >, < D, +, 0 >, m >$. It is easy to show that $< \mathcal{F}', \mathcal{I} >, [t'_1, t'_2] \models_{hdc} \Gamma$ since the interpretations of HDC are independent of D_1. □

Theorem 6 (Completeness). *The proof system of HDC is complete, i.e. $\models_{hdc} \phi$ implies $\vdash_{hdc} \phi$.*

Proof. Suppose $\models_{hdc} \phi$ but $\not\vdash_{hdc} \phi$. So $\{\neg\phi\}$ is consistent with respect to the proof system of HDC. By Theorem 5, there exists a model $< \mathcal{F}, \mathcal{I} >$ and an interval $[t_1, t_2]$ such that $< \mathcal{F}, \mathcal{I} >, [t_1, t_2] \models_{hdc} \neg\phi$. This contradicts $\models_{hdc} \phi$. Hence, $\vdash_{hdc} \phi$. □

5 Discussion

In order to develop a DC-based programming theory, a higher-order duration calculus has been established in [2]. In this paper, we investigate the logic properties of HDC. Especially, we proved that HDC is complete on abstract domains by reducing HDC to a complete first-order two-sorted interval temporal logic.

In the literature of DC, there are two completeness results. One is on abstract domains (see [8]). Unfortunately it requires ω-rule. The other is on real

domain (see [10]), but it is a relative completeness, i.e. it is assumed that all valid formulae of real arithmetic and interval temporal logic are provable in DC. Up to now, no one find a relation between these two completeness results.

If we give another relative completeness of HDC, i.e. if $\models_{hdc} \phi$, then $\Gamma_R \vdash_{hdc} \phi$, where Γ_R stands for all valid real formulae, then we can show that if interval temporal logic is complete on real domain w.r.t. the assumption that all valid real formulae are provable, then completeness of HDC on real domain under the same assumption can be proved with the technique developed in this paper. This conclusion can be applied to other variants of DC too. But how to prove the relative completeness of temporal logic on real domain is still an open problem.

Acknowledgements

The author sincerely give his thanks to his supervisor, Prof. Zhou Chaochen for his instructions and guidance, many inspiring discussions and many suggestions which improved the presentation of this paper. The author is indebted to Dr. Dimitar P. Guelev for his idea to reduce higher-order logic to first-order one.

References

1. M Abadi. The power of temporal proofs. *Theoretical Computer Science*, 1989, 65: 35-83, *Corrigendum in TCS 70 (1990), page 275*
2. Zhou Chaochen, Dimitar P. Guelev and Zhan Naijun. A higher-order duration calculus. UNU/IIST Report No. 167, UNU/IIST, P.O. Box 3058, Macau, July, 1999.
3. Zhou Chaochen, C.A.R. Hoare, and A.P. Ravn. A calculus of durations. *Information Processing Letters*, 1991,40(5):269–276.
4. Zhou Chaochen and Li Xiaoshan. A mean value calculus of durations. In *A Classical Mind: Essays in Honour of C.A.R. Hoare*, Prentice Hall, 1994, 431–451.
5. B. Dutertre. On first order interval temporal logic. Report no. CSD-TR-94-3, Department of Computer Science, Royal Holloway, University of London, Eghan, Surrey TW20 0EX, England, 1995.
6. R.L. Epstein. *The Semantic Foundations of Logic: Predicate Logic*, Oxford University Press, Oxford, UK, 1994.
7. J.W. Garson. Quantification in modal logic. In *Handbook of Philosophical Logic*, D. Gabbay and F. Guenther (Eds), Reidel, 1984, (II):249-307.
8. Dimitar P. Guelev. A calculus of durations on abstract domains: completeness and extensions. UNU/IIST Report No. 139, UNU/IIST, P.O. Box 3058, Macau, May, 1998.
9. Dimitar P. Guelev. Quantification over States in Duration Calculus. August, 1998.
10. M.R. Hansen and Zhou Chaochen. Semantics and completeness of duration calculus. In *Real-Time: Theory in Practice*, Springer-Verlag, 1992,LNCS 600, 209–225.
11. M.R. Hansen and Zhou Chaochen. Duration calculus: logical foundations. *Formal Aspects of Computing*,1997, 9:283-330.
12. Zhan Naijun. Completeness of higher-order duration calculus. UNU/IIST Report No.175, UNU/IIST, P.O. Box 3058, Macau, August, 1999.

Equational Termination by Semantic Labelling

Hitoshi Ohsaki[1], Aart Middeldorp[2], and Jürgen Giesl[3]

[1] Computer Science Division, Electrotechnical Laboratory
Tsukuba 305-8568, Japan
ohsaki@etl.go.jp

[2] Institute of Information Sciences and Electronics
University of Tsukuba, Tsukuba 305-8573, Japan
ami@is.tsukuba.ac.jp

[3] Computer Science Department
University of New Mexico, Albuquerque, NM 87131, USA
giesl@cs.unm.edu

Abstract. Semantic labelling is a powerful tool for proving termination of term rewrite systems. The usefulness of the extension to equational term rewriting described in Zantema [24] is however rather limited. In this paper we introduce a stronger version of *equational semantical labelling*, parameterized by three choices: (1) the order on the underlying algebra (partial order vs. quasi-order), (2) the relation between the algebra and the rewrite system (model vs. quasi-model), and (3) the labelling of the function symbols appearing in the equations (forbidden vs. allowed). We present soundness and completeness results for the various instantiations and analyze the relationships between them. Applications of our equational semantic labelling technique include a short proof of the main result of Ferreira *et al.* [7]—the correctness of a version of dummy elimination for AC-rewriting which completely removes the AC-axioms—and an extension of Zantema's distribution elimination technique [23] to the equational setting.

1 Introduction

This paper is concerned with termination of equational term rewrite systems. Termination of ordinary term rewrite systems has been extensively studied and several powerful methods for establishing termination are available (e.g. [1,4,21]). For equational term rewriting much less is known, although in recent years significant progress has been made with respect to AC-termination, i.e., termination of equational rewrite systems where the set of equations consists of the associativity and commutativity axioms $\mathrm{AC}(f) = \{f(f(x,y),z) \approx f(x,f(y,z)), f(x,y) \approx f(y,x)\}$ for (some of) the binary function symbols occurring in the rewrite rules. An early paper on termination of equational rewriting is Jouannaud and Muñoz [11]. In that paper sufficient conditions are given for reducing termination of an equational term rewrite system to termination of its underlying term rewrite system. In another early paper (Ben Cherifa and Lescanne [2]) a characterization of the polynomials is given that can be used in a polynomial

P. Clote and H. Schwichtenberg (Eds.): CSL 2000, LNCS 1862, pp. 457–471, 2000.

interpretation proof of AC-termination. In more recent papers [12,19,20,21] syntactic methods like the well-known recursive path order for proving termination of rewriting are extended to AC-rewriting. Marché and Urbain [14] extended the powerful dependency pair technique of Arts and Giesl [1] to AC-rewriting. In [6,7] two extensions of dummy elimination ([8]) to equational rewriting are presented. In [15] the type introduction technique of Zantema [23] is extended to equational term rewriting.

In this paper we extend another technique of Zantema to equational term rewriting. By labelling function symbols according to the semantics of the rewrite system, semantic labelling ([24]) transforms a rewrite system into another rewrite system with the same termination behaviour. The aim is to obtain a transformed rewrite system where termination is easier to establish. The strength of semantic labelling is amply illustrated in [16,24]. Here we present powerful extensions of semantic labelling to equational rewriting and analyze their soundness and completeness. Our equational semantic labelling yields a short correctness proof of a version of dummy elimination for AC-rewriting. This result of Ferreira *et al.* was obtained in [7] by considerably more complicated arguments. Another application of our technique is the extension of some of the results of Zantema [23] concerning distribution elimination to the AC case.

2 Preliminaries

Familiarity with the basics of term rewriting ([3]) is assumed. An *equational system* (ES for short) consists of a signature $\mathcal{F}$ and a set $\mathcal{E}$ of equations between terms in $\mathcal{T}(\mathcal{F},\mathcal{V})$. We write $s \to_{\mathcal{E}} t$ if there exist an equation $l \approx r$ in $\mathcal{E}$, a substitution σ, and a context C such that $s = C[l\sigma]$ and $t = C[r\sigma]$. The symmetric closure of $\to_{\mathcal{E}}$ is denoted by $\vdash\!\dashv_{\mathcal{E}}$ and the transitive reflexive closure of $\vdash\!\dashv_{\mathcal{E}}$ by $\sim_{\mathcal{E}}$. A rewrite rule is an equation $l \approx r$ such that l is not a variable and variables which occur in r also occur in l. Rewrite rules $l \approx r$ are written as $l \to r$. A *term rewrite system* (TRS for short) is an ES with the property that all its equations are rewrite rules. An *equational term rewrite system* (ETRS for short) $\mathcal{R}/\mathcal{E}$ consists of a TRS $\mathcal{R}$ and an ES $\mathcal{E}$ over the same signature. We write $s \to_{\mathcal{R}/\mathcal{E}} t$ if there exist terms s' and t' such that $s \sim_{\mathcal{E}} s' \to_{\mathcal{R}} t' \sim_{\mathcal{E}} t$. Similar to ordinary term rewrite systems, an ETRS is called *terminating* if there does not exist an infinite $\to_{\mathcal{R}/\mathcal{E}}$ reduction.

Let $\mathcal{F}$ be a signature and $\mathcal{A} = (A, \{f_{\mathcal{A}}\}_{f\in\mathcal{F}})$ an $\mathcal{F}$-algebra equipped with a quasi-order (i.e., a reflexive and transitive relation) $\succsim$ on its (non-empty) carrier A. For any variable assignment $\alpha\colon \mathcal{V} \to A$ we define the term evaluation $[\alpha]_{\mathcal{A}}\colon \mathcal{T}(\mathcal{F},\mathcal{V}) \to A$ inductively by $[\alpha]_{\mathcal{A}}(x) = \alpha(x)$ and $[\alpha]_{\mathcal{A}}(f(t_1,\ldots,t_n)) = f_{\mathcal{A}}([\alpha]_{\mathcal{A}}(t_1),\ldots,[\alpha]_{\mathcal{A}}(t_n))$ for $x \in \mathcal{V}$, $f \in \mathcal{F}$, and $t_1,\ldots,t_n \in \mathcal{T}(\mathcal{F},\mathcal{V})$. If $\mathcal{A}$ is clear from the context, then we often write $[\alpha]$ instead of $[\alpha]_{\mathcal{A}}$. We say that $\mathcal{A}$ is *monotone* if the algebra operations of $\mathcal{A}$ are monotone with respect to $\succsim$ in all coordinates, i.e., if $f \in \mathcal{F}$ has arity $n \geqslant 1$ then $f_{\mathcal{A}}(a_1,\ldots,a_i,\ldots,a_n) \succsim f_{\mathcal{A}}(a_1,\ldots,b,\ldots,a_n)$ for all $a_1,\ldots,a_n,b \in A$ and $i \in \{1,\ldots,n\}$ with $a_i \succsim b$. An ETRS $\mathcal{R}/\mathcal{E}$ over a signature $\mathcal{F}$ is *compatible* with a monotone $\mathcal{F}$-algebra

$(\mathcal{A}, \succsim)$ if $l \succsim_{\mathcal{A}} r$ for every rewrite rule $l \to r \in \mathcal{R}$ and $l \sim_{\mathcal{A}} r$ for every equation $l \approx r \in \mathcal{E}$. Here the relation $\succsim_{\mathcal{A}}$ is defined by $s \succsim_{\mathcal{A}} t$ if $[\alpha]_{\mathcal{A}}(s) \succsim [\alpha]_{\mathcal{A}}(t)$ for every assignment α and $\sim_{\mathcal{A}}$ is the equivalence relation induced by $\succsim_{\mathcal{A}}$. If $\mathcal{R}/\mathcal{E}$ and $(\mathcal{A}, \succsim)$ are compatible, we also say that $(\mathcal{A}, \succsim)$ is a *quasi-model* of $\mathcal{R}/\mathcal{E}$. We call $(\mathcal{A}, \succsim)$ a *model* of $\mathcal{R}/\mathcal{E}$ if $l \sim_{\mathcal{A}} r$ for all $l \to r \in \mathcal{R}$ and $l \approx r \in \mathcal{E}$.

A TRS $\mathcal{R}$ is *precedence terminating* if there exists a well-founded order $\sqsupset$ on its signature $\mathcal{F}$ such that $\mathrm{root}(l) \sqsupset f$ for every rule $l \to r \in \mathcal{R}$ and every function symbol f occurring in r. Precedence terminating TRSs are terminating ([16]). The next lemma states that this remains true in the presence of AC-axioms.

Lemma 1. *Let $\mathcal{R}/\mathcal{E}$ be an ETRS over a signature $\mathcal{F}$ such that $\mathcal{E} = \bigcup_{f \in \mathcal{G}} \mathrm{AC}(f)$ for some subset $\mathcal{G}$ of $\mathcal{F}$. If $\mathcal{R}$ is precedence terminating then $\mathcal{R}/\mathcal{E}$ is terminating.*

Proof. By definition there is a well-founded order $\sqsupset$ on $\mathcal{F}$ such that $\mathrm{root}(l) \sqsupset f$ for every rule $l \to r \in \mathcal{R}$ and every function symbol f occurring in r. Any AC-compatible recursive path order induced by $\sqsupset$ that is defined on terms with variables (e.g. [13,19]) orients the rules of $\mathcal{R}$ from left to right. (The complicated case in which two terms with equal root symbols in $\mathcal{G}$ have to be compared never arises due to the assumption on $\sqsupset$.) We conclude that $\mathcal{R}/\mathcal{E}$ is terminating. □

3 Semantic Labelling for Equational Rewriting

In this section we present our equational semantic labelling framework by appropriately extending the definitions of Zantema [24] for ordinary semantic labelling.

Definition 1. *Let $\mathcal{F}$ be a signature and $\mathcal{A}$ an $\mathcal{F}$-algebra. A* labelling *L for $\mathcal{F}$ consists of sets of labels $L_f \subseteq A$ for every $f \in \mathcal{F}$. The* labelled signature *$\mathcal{F}_{\mathrm{lab}}$ consists of n-ary function symbols f_a for every n-ary function symbol $f \in \mathcal{F}$ and label $a \in L_f$ together with all function symbols $f \in \mathcal{F}$ such that $L_f = \varnothing$. A* labelling *ℓ for $\mathcal{A}$ consists of a labelling L for the signature $\mathcal{F}$ together with mappings $\ell_f \colon A^n \to L_f$ for every n-ary function symbol $f \in \mathcal{F}$ with $L_f \neq \varnothing$. If $\mathcal{A}$ is equipped with a quasi-order $\succsim$ then the labelling is said to be* monotone *if its labelling functions ℓ_f are monotone (with respect to $\succsim$) in all arguments.*

Definition 2. *Let $\mathcal{R}/\mathcal{E}$ be an ETRS over a signature $\mathcal{F}$, $(\mathcal{A}, \succsim)$ an $\mathcal{F}$-algebra, and ℓ a labelling for $\mathcal{A}$. For every assignment α we inductively define a labelling function lab_α from $\mathcal{T}(\mathcal{F}, \mathcal{V})$ to $\mathcal{T}(\mathcal{F}_{\mathrm{lab}}, \mathcal{V})$: $\mathrm{lab}_\alpha(t) = t$ if $t \in \mathcal{V}$ and $\mathrm{lab}_\alpha(t) = f_{\ell_f([\alpha](t_1), \ldots, [\alpha](t_n))}(\mathrm{lab}_\alpha(t_1), \ldots, \mathrm{lab}_\alpha(t_n))$ if $t = f(t_1, \ldots, t_n)$. We define TRSs $\mathcal{R}_{\mathrm{lab}}$, $\mathrm{Dec}(\mathcal{F}, \succ)$ and ESs $\mathcal{E}_{\mathrm{lab}}$, $\mathrm{Eq}(\mathcal{F}, \sim)$ over the signature $\mathcal{F}_{\mathrm{lab}}$ as follows:*

$$\begin{aligned}
\mathcal{R}_{\mathrm{lab}} &= \{\, \mathrm{lab}_\alpha(l) \to \mathrm{lab}_\alpha(r) \mid l \to r \in \mathcal{R} \text{ and } \alpha \colon \mathcal{V} \to A \,\}, \\
\mathcal{E}_{\mathrm{lab}} &= \{\, \mathrm{lab}_\alpha(l) \approx \mathrm{lab}_\alpha(r) \mid l \approx r \in \mathcal{E} \text{ and } \alpha \colon \mathcal{V} \to A \,\}, \\
\mathrm{Dec}(\mathcal{F}, \succ) &= \{ f_a(x_1, \ldots, x_n) \to f_b(x_1, \ldots, x_n) \mid f \in \mathcal{F},\ a, b \in L_f,\ a \succ b \}, \\
\mathrm{Eq}(\mathcal{F}, \sim) &= \{ f_a(x_1, \ldots, x_n) \approx f_b(x_1, \ldots, x_n) \mid f \in \mathcal{F},\ a, b \in L_f,\ a \sim b,\ a \neq b \}.
\end{aligned}$$

The purpose of the condition $a \neq b$ in the definition of $\mathrm{Eq}(\mathcal{F}, \sim)$ is to exclude trivial equations. When the signature $\mathcal{F}$ and the quasi-order $\succsim$ can be inferred from the context we just write Dec *and* Eq. *We write $\overline{\mathcal{R}}$ for the union of $\mathcal{R}_{\mathrm{lab}}$ and* Dec *and $\overline{\mathcal{E}}$ for the union of $\mathcal{E}_{\mathrm{lab}}$ and* Eq.

The next theorem states our first equational semantic labelling result.

Theorem 1. *Let $\mathcal{R}/\mathcal{E}$ be an ETRS over a signature $\mathcal{F}$, $(\mathcal{A}, \succsim)$ a monotone $\mathcal{F}$-algebra, and ℓ a monotone labelling for $\mathcal{A}$. If $\mathcal{A}$ is a quasi-model of $\mathcal{R}/\mathcal{E}$ and $\overline{\mathcal{R}}/\overline{\mathcal{E}}$ is terminating then $\mathcal{R}/\mathcal{E}$ is terminating.*

Proof. We show that for all terms $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ and assignments α we have

1. if $s \to_{\mathcal{R}} t$ then $\mathrm{lab}_\alpha(s) \sim_{\overline{\mathcal{E}}} \cdot \xrightarrow{+}_{\overline{\mathcal{R}}} \mathrm{lab}_\alpha(t)$,
2. if $s \vdash\!\!\!\rightarrow_{\mathcal{E}} t$ then $\mathrm{lab}_\alpha(s) \sim_{\overline{\mathcal{E}}} \mathrm{lab}_\alpha(t)$.

Suppose $s = C[l\sigma]$ and $t = C[r\sigma]$ for some rewrite rule $l \to r \in \mathcal{R}$, context C, and substitution σ. We show (1) by induction on C. If $C = \Box$ then $\mathrm{lab}_\alpha(s) = \mathrm{lab}_\alpha(l\sigma)$ and $\mathrm{lab}_\alpha(t) = \mathrm{lab}_\alpha(r\sigma)$. Define the assignment $\beta = [\alpha]_{\mathcal{A}} \circ \sigma$ and the substitution $\tau = \mathrm{lab}_\alpha \circ \sigma$ (i.e., σ is applied first). An easy induction proof (e.g. [23, Lemma 2]) reveals that $\mathrm{lab}_\alpha(l\sigma) = \mathrm{lab}_\beta(l)\tau$ and $\mathrm{lab}_\alpha(r\sigma) = \mathrm{lab}_\beta(r)\tau$. By definition $\mathrm{lab}_\beta(l) \to \mathrm{lab}_\beta(r) \in \mathcal{R}_{\mathrm{lab}}$ and hence $\mathrm{lab}_\alpha(s) = \mathrm{lab}_\beta(l)\tau \to_{\mathcal{R}_{\mathrm{lab}}} \mathrm{lab}_\beta(r)\tau = \mathrm{lab}_\alpha(t)$. For the induction step, let $C = f(u_1, \ldots, C', \ldots, u_n)$. The induction hypothesis yields $\mathrm{lab}_\alpha(C'[l\sigma]) \sim_{\overline{\mathcal{E}}} \cdot \xrightarrow{+}_{\overline{\mathcal{R}}} \mathrm{lab}_\alpha(C'[r\sigma])$. Because $\mathcal{A}$ is a quasi-model of $\mathcal{R}/\mathcal{E}$ and $C'[l\sigma] \to_{\mathcal{R}} C'[r\sigma]$, we have $[\alpha]_{\mathcal{A}}(C'[l\sigma]) \succsim [\alpha]_{\mathcal{A}}(C'[r\sigma])$. Let

$$a = \ell_f([\alpha]_{\mathcal{A}}(u_1), \ldots, [\alpha]_{\mathcal{A}}(C'[l\sigma]), \ldots, [\alpha]_{\mathcal{A}}(u_n))$$

and

$$b = \ell_f([\alpha]_{\mathcal{A}}(u_1), \ldots, [\alpha]_{\mathcal{A}}(C'[r\sigma]), \ldots, [\alpha]_{\mathcal{A}}(u_n)).$$

Monotonicity of the labelling function ℓ_f yields $a \succsim b$. We distinguish two cases. If $a \succ b$ then

$$\begin{array}{lll} \mathrm{lab}_\alpha(s) & \sim_{\overline{\mathcal{E}}} \cdot \xrightarrow{+}_{\overline{\mathcal{R}}} & f_a(\mathrm{lab}_\alpha(u_1), \ldots, \mathrm{lab}_\alpha(C'[r\sigma]), \ldots, \mathrm{lab}_\alpha(u_n)) \\ & \to_{\mathrm{Dec}} & f_b(\mathrm{lab}_\alpha(u_1), \ldots, \mathrm{lab}_\alpha(C'[r\sigma]), \ldots, \mathrm{lab}_\alpha(u_n)) \\ & = & \mathrm{lab}_\alpha(t). \end{array}$$

If $a \sim b$ then

$$\begin{array}{lll} \mathrm{lab}_\alpha(s) & \vdash\!\!\!\rightarrow^{=}_{\mathrm{Eq}} & f_b(\mathrm{lab}_\alpha(u_1), \ldots, \mathrm{lab}_\alpha(C'[l\sigma]), \ldots, \mathrm{lab}_\alpha(u_n)) \\ & \sim_{\overline{\mathcal{E}}} \cdot \xrightarrow{+}_{\overline{\mathcal{R}}} & f_b(\mathrm{lab}_\alpha(u_1), \ldots, \mathrm{lab}_\alpha(C'[r\sigma]), \ldots, \mathrm{lab}_\alpha(u_n)) \\ & = & \mathrm{lab}_\alpha(t). \end{array}$$

Here $\vdash\!\!\!\rightarrow^{=}_{\mathrm{Eq}}$ denotes $\vdash\!\!\!\rightarrow_{\mathrm{Eq}} \cup =$. Since $\sim_{\overline{\mathcal{E}}} \cdot \xrightarrow{+}_{\overline{\mathcal{R}}} \cdot \to_{\mathrm{Dec}} \subseteq \sim_{\overline{\mathcal{E}}} \cdot \xrightarrow{+}_{\overline{\mathcal{R}}}$ and $\vdash\!\!\!\rightarrow^{=}_{\mathrm{Eq}} \cdot \sim_{\overline{\mathcal{E}}} \cdot \xrightarrow{+}_{\overline{\mathcal{R}}} \subseteq \sim_{\overline{\mathcal{E}}} \cdot \xrightarrow{+}_{\overline{\mathcal{R}}}$, in both cases we obtain the desired $\mathrm{lab}_\alpha(s) \sim_{\overline{\mathcal{E}}} \cdot \xrightarrow{+}_{\overline{\mathcal{R}}} \mathrm{lab}_\alpha(t)$.

The proof of (2) follows along the same lines. In the induction step we have $[\alpha]_{\mathcal{A}}(C'[l\sigma]) \sim [\alpha]_{\mathcal{A}}(C'[r\sigma])$. Monotonicity of ℓ_f yields both $a \succsim b$ and $b \succsim a$. Hence $a \sim b$ and thus

$$\begin{array}{lll} \mathrm{lab}_\alpha(s) = & f_a(\mathrm{lab}_\alpha(u_1), \ldots, \mathrm{lab}_\alpha(C'[l\sigma]), \ldots, \mathrm{lab}_\alpha(u_n)) \\ \vdash\!\!\!\rightarrow^{=}_{\mathrm{Eq}} & f_b(\mathrm{lab}_\alpha(u_1), \ldots, \mathrm{lab}_\alpha(C'[l\sigma]), \ldots, \mathrm{lab}_\alpha(u_n)) \\ \sim_{\overline{\mathcal{E}}} & f_b(\mathrm{lab}_\alpha(u_1), \ldots, \mathrm{lab}_\alpha(C'[r\sigma]), \ldots, \mathrm{lab}_\alpha(u_n)) \\ = & \mathrm{lab}_\alpha(t) \end{array}$$

by the definition of Eq and the induction hypothesis.

From (1) and (2) it follows that any infinite $\mathcal{R}/\mathcal{E}$-rewrite sequence gives rise to an infinite $\overline{\mathcal{R}}/\overline{\mathcal{E}}$-rewrite sequence. □

The converse of the above theorem does not hold. Consider the terminating ETRS $\mathcal{R}/\mathcal{E}$ with $\mathcal{R} = \varnothing$ and $\mathcal{E} = \{\mathsf{f}(\mathsf{a}) \approx \mathsf{a}\}$. Let $\mathcal{A}$ be the algebra over the carrier $\{0,1\}$ with $1 \succ 0$ and operations $\mathsf{f}_{\mathcal{A}}(x) = x$ for all $x \in \{0,1\}$ and $\mathsf{a}_{\mathcal{A}} = 1$. Note that $\mathcal{A}$ is a (quasi-)model of $\mathcal{R}/\mathcal{E}$. By letting ℓ_{f} be the identity function and by choosing $L_{\mathsf{a}} = \varnothing$, we obtain the labelled ETRS $\overline{\mathcal{R}}/\overline{\mathcal{E}}$ with $\mathcal{R}_{\mathrm{lab}} = \varnothing$, $\mathrm{Dec} = \{\mathsf{f}_1(x) \to \mathsf{f}_0(x)\}$, $\mathcal{E}_{\mathrm{lab}} = \{\mathsf{f}_1(\mathsf{a}) \approx \mathsf{a}\}$, and $\mathrm{Eq} = \varnothing$. The ETRS $\overline{\mathcal{R}}/\overline{\mathcal{E}}$ is not terminating: $\mathsf{a} \sim_{\mathcal{E}_{\mathrm{lab}}} \mathsf{f}_1(\mathsf{a}) \to_{\mathrm{Dec}} \mathsf{f}_0(\mathsf{a}) \sim_{\mathcal{E}_{\mathrm{lab}}} \mathsf{f}_0(\mathsf{f}_1(\mathsf{a})) \to_{\mathrm{Dec}} \cdots$ Nevertheless, in this example there are no infinite $\overline{\mathcal{R}}/\overline{\mathcal{E}}$-rewrite sequences that contain infinitely many $\mathcal{R}_{\mathrm{lab}}/\overline{\mathcal{E}}$-steps, which is known as the *relative termination* (Geser [10]) of $\mathcal{R}_{\mathrm{lab}}/\overline{\mathcal{E}}$ with respect to Dec. It is not difficult to show that under the assumptions of Theorem 1 termination of $\mathcal{R}/\mathcal{E}$ is equivalent to relative termination of $\mathcal{R}_{\mathrm{lab}}/\overline{\mathcal{E}}$ with respect to Dec.

Zantema [24] showed the necessity of the inclusion of Dec in $\overline{\mathcal{R}}$ for the correctness of Theorem 1 (with $\mathcal{E} = \varnothing$) by means of the TRS $\mathcal{R} = \{\mathsf{f}(\mathsf{g}(x)) \to \mathsf{g}(\mathsf{g}(\mathsf{f}(\mathsf{f}(x))))\}$, the algebra $\mathcal{A}$ over the carrier $\{0,1\}$ with operations $\mathsf{f}_{\mathcal{A}}(x) = 1$ and $\mathsf{g}_{\mathcal{A}}(x) = 0$ for all $x \in \{0,1\}$, and the order $1 \succ 0$. By labelling f with the value of its argument, we obtain the TRS $\mathcal{R}_{\mathrm{lab}} = \{\mathsf{f}_0(\mathsf{g}(x)) \to \mathsf{g}(\mathsf{g}(\mathsf{f}_1(\mathsf{f}_0(x)))), \mathsf{f}_0(\mathsf{g}(x)) \to \mathsf{g}(\mathsf{g}(\mathsf{f}_1(\mathsf{f}_1(x))))\}$ which is compatible with the recursive path order with precedence $\mathsf{f}_0 \sqsupset \mathsf{f}_1, \mathsf{g}$. However, $\mathcal{R}$ is not terminating: $\mathsf{f}(\mathsf{f}(\mathsf{g}(x))) \to \mathsf{f}(\mathsf{g}(\mathsf{g}(\mathsf{f}(\mathsf{f}(x))))) \to \mathsf{g}(\mathsf{g}(\mathsf{f}(\mathsf{f}(\mathsf{g}(\mathsf{f}(\mathsf{f}(x))))))) \to \cdots$

The inclusion of Eq in $\overline{\mathcal{E}}$ is also essential for the correctness of Theorem 1. Consider the ETRS $\mathcal{R}/\mathcal{E}$ with $\mathcal{R} = \{\mathsf{f}(\mathsf{a},\mathsf{b},x) \to \mathsf{f}(x,x,x), \mathsf{g}(x,y) \to x, \mathsf{g}(x,y) \to y\}$ and $\mathcal{E} = \varnothing$. Let $\mathcal{A}$ be the algebra over the carrier $\{0,1\}$ with $0 \sim 1$ and operations $\mathsf{f}_{\mathcal{A}}(x,y,z) = 1$, $\mathsf{g}_{\mathcal{A}}(x,y) = 0$, $\mathsf{a}_{\mathcal{A}} = 0$, and $\mathsf{b}_{\mathcal{A}} = 1$. We label function symbol f as follows: $\ell_{\mathsf{f}}(x,y,z) = 0$ if $x = y$ and $\ell_{\mathsf{f}}(x,y,z) = 1$ if $x \neq y$. Note that $\mathcal{A}$ is a quasi-model for $\mathcal{R}/\mathcal{E}$ and ℓ_{f} is trivially monotone. We have $\mathcal{R}_{\mathrm{lab}} = \{\mathsf{f}_1(\mathsf{a},\mathsf{b},x) \to \mathsf{f}_0(x,x,x), \mathsf{g}(x,y) \to x, \mathsf{g}(x,y) \to y\}$, $\mathrm{Dec} = \varnothing$, and $\mathcal{E}_{\mathrm{lab}} = \varnothing$. Termination of $\overline{\mathcal{R}}$ is easily shown. It is well-known (Toyama [22]) that $\mathcal{R}$ is not terminating. Note that in this example $\mathrm{Eq} = \{\mathsf{f}_0(x,y,z) \approx \mathsf{f}_1(x,y,z)\}$ and hence $\overline{\mathcal{R}}/\overline{\mathcal{E}}$ is not terminating.

Finally, both monotonicity requirements are essential. Consider the TRS $\mathcal{R} = \{\mathsf{f}(\mathsf{g}(\mathsf{a})) \to \mathsf{f}(\mathsf{g}(\mathsf{b})), \mathsf{b} \to \mathsf{a}\}$. Let $\mathcal{A}$ be the algebra over the carrier $\{0,1\}$ with $1 \succ 0$ and operations $\mathsf{f}_{\mathcal{A}}(x) = 0$, $\mathsf{g}_{\mathcal{A}}(x) = 1 - x$, $\mathsf{a}_{\mathcal{A}} = 0$, and $\mathsf{b}_{\mathcal{A}} = 1$. We have $l \succsim_{\mathcal{A}} r$ for both rules $l \to r \in \mathcal{R}$. If $\ell_{\mathsf{f}}(x) = x$ then we obtain the TRS $\overline{\mathcal{R}} = \{\mathsf{f}_1(\mathsf{g}(\mathsf{a})) \to \mathsf{f}_0(\mathsf{g}(\mathsf{b})), \mathsf{b} \to \mathsf{a}, \mathsf{f}_1(x) \to \mathsf{f}_0(x)\}$ which is compatible with the recursive path order with precedence $\mathsf{f}_1 \sqsupset \mathsf{f}_0, \mathsf{g}$ and $\mathsf{f}_1 \sqsupset \mathsf{b} \sqsupset \mathsf{a}$. However, $\mathcal{R}$ is not terminating. Note that $\mathsf{g}_{\mathcal{A}}$ is not monotone. Next consider the algebra $\mathcal{B}$ over the carrier $\{0,1\}$ with $1 \succ 0$ and operations $\mathsf{f}_{\mathcal{B}}(x) = 0$, $\mathsf{g}_{\mathcal{B}}(x) = x$, $\mathsf{a}_{\mathcal{B}} = 0$, and $\mathsf{b}_{\mathcal{B}} = 1$. If $\ell_{\mathsf{f}}(x) = 1 - x$ then we obtain the same TRS $\overline{\mathcal{R}}$ as before. Note that now ℓ_{f} is not monotone.

If the algebra $\mathcal{A}$ is a model of the ETRS $\mathcal{R}/\mathcal{E}$ then (similar to ordinary semantic labelling [24]) we can dispense with Dec. Moreover, in this case the converse of Theorem 1 also holds. This is expressed in the next theorem.

Theorem 2. *Let $\mathcal{R}/\mathcal{E}$ be an ETRS over a signature $\mathcal{F}$, $(\mathcal{A}, \succsim)$ a monotone $\mathcal{F}$-algebra, and ℓ a monotone labelling for $\mathcal{A}$. If $\mathcal{A}$ is a model of $\mathcal{R}/\mathcal{E}$ then termination of $\mathcal{R}_{\mathrm{lab}}/\overline{\mathcal{E}}$ is equivalent to termination of $\mathcal{R}/\mathcal{E}$.*

Proof. The following statements are obtained by a straightforward modification of the proof of Theorem 1:

1. if $s \to_{\mathcal{R}} t$ then $\mathrm{lab}_\alpha(s) \sim_{\overline{\mathcal{E}}} \cdot \to_{\mathcal{R}_{\mathrm{lab}}} \mathrm{lab}_\alpha(t)$,
2. if $s \vdash\!\!\!\to_{\mathcal{E}} t$ then $\mathrm{lab}_\alpha(s) \sim_{\overline{\mathcal{E}}} \mathrm{lab}_\alpha(t)$.

Note that since $\mathcal{A}$ is a model we have $[\alpha]_{\mathcal{A}}(C'[l\sigma]) \sim [\alpha]_{\mathcal{A}}(C'[r\sigma])$ and hence $a \sim b$ in the induction step. This explains why there is no need for Dec. So termination of $\mathcal{R}_{\mathrm{lab}}/\overline{\mathcal{E}}$ implies termination of $\mathcal{R}/\mathcal{E}$. The converse also holds; eliminating all labels in an infinite $\mathcal{R}_{\mathrm{lab}}/\overline{\mathcal{E}}$-rewrite sequence yields an infinite $\mathcal{R}/\mathcal{E}$-rewrite sequence (because there are infinitely many $\mathcal{R}_{\mathrm{lab}}$-steps). □

If the quasi-model $\mathcal{A}$ in Theorem 1 is equipped with a partial order (i.e., a reflexive, transitive, and anti-symmetric relation) $\succeq$ instead of a quasi-order $\succsim$ then we can dispense with Eq.

Theorem 3. *Let $\mathcal{R}/\mathcal{E}$ be an ETRS over a signature $\mathcal{F}$, $(\mathcal{A}, \succeq)$ a monotone $\mathcal{F}$-algebra, and ℓ a monotone labelling for $\mathcal{A}$. If $\mathcal{A}$ is a quasi-model of $\mathcal{R}/\mathcal{E}$ and $\overline{\mathcal{R}}/\mathcal{E}_{\mathrm{lab}}$ is terminating then $\mathcal{R}/\mathcal{E}$ is terminating.*

Proof. The proof of Theorem 1 applies; because the equivalence associated with a partial order is the identity relation we have $\mathrm{Eq} = \varnothing$. □

The first example in this section shows that the converse of Theorem 3 does not hold. Combining the preceding two theorems yields the following result.

Corollary 1. *Let $\mathcal{R}/\mathcal{E}$ be an ETRS over a signature $\mathcal{F}$, $(\mathcal{A}, \succeq)$ a monotone $\mathcal{F}$-algebra, and ℓ a monotone labelling for $\mathcal{A}$. If $\mathcal{A}$ is a model of $\mathcal{R}/\mathcal{E}$ then termination of $\mathcal{R}_{\mathrm{lab}}/\mathcal{E}_{\mathrm{lab}}$ is equivalent to termination of $\mathcal{R}/\mathcal{E}$.* □

Note that if the pair $(\mathcal{A}, \succeq)$ is a model of $\mathcal{R}/\mathcal{E}$ then so is $(\mathcal{A}, =)$. Since in this case monotonicity of both the algebra operations and the labelling functions is trivially satisfied, we can rephrase the above corollary as follows.

Corollary 2. *Let $\mathcal{R}/\mathcal{E}$ be an ETRS over a signature $\mathcal{F}$, $\mathcal{A}$ an $\mathcal{F}$-algebra, and ℓ a labelling for $\mathcal{A}$. If $\mathcal{A}$ is a model of $\mathcal{R}/\mathcal{E}$ then termination of $\mathcal{R}_{\mathrm{lab}}/\mathcal{E}_{\mathrm{lab}}$ is equivalent to termination of $\mathcal{R}/\mathcal{E}$.* □

Note that the unspecified quasi-order is assumed to be the identity relation, so model here means $l =_{\mathcal{A}} r$ for all rules $l \to r \in \mathcal{R}$ and all equations $l \approx r \in \mathcal{E}$.

Let us conclude this section by illustrating the power of equational semantic labelling on a concrete example. Consider the ETRS $\mathcal{R}/\mathcal{E}$ with $\mathcal{R} = \{x - \mathsf{0} \to x,\ \mathsf{s}(x) - \mathsf{s}(y) \to x - y,\ \mathsf{0} \div \mathsf{s}(y) \to \mathsf{0},\ \mathsf{s}(x) \div \mathsf{s}(y) \to \mathsf{s}((x - y) \div \mathsf{s}(y))\}$ and $\mathcal{E} = \{(x \div y) \div z \approx (x \div z) \div y\}$. Let $\mathcal{A}$ be the algebra with carrier $\mathbb{N}$, standard order $\geqslant$, and operations $\mathsf{0}_{\mathcal{A}} = 0$, $\mathsf{s}_{\mathcal{A}}(x) = x + 1$, and $x -_{\mathcal{A}} y = x \div_{\mathcal{A}} y = x$. This algebra is a quasi-model of $\mathcal{R}/\mathcal{E}$. If $\ell_{\div}(x, y) = x$ then we have $\mathcal{R}_{\mathrm{lab}} = \{x - \mathsf{0} \to x,\ \mathsf{s}(x) - \mathsf{s}(y) \to x - y,\ \mathsf{0} \div_0 \mathsf{s}(y) \to \mathsf{0}\} \cup \{\mathsf{s}(x) \div_{n+1} \mathsf{s}(y) \to \mathsf{s}((x - y) \div_n \mathsf{s}(y)) \mid n \geqslant\}$,

$\mathrm{Dec} = \{x \div_m y \to x \div_n y \mid m > n\}$, and $\mathcal{E}_{\mathrm{lab}} = \{(x \div_n y) \div_n z \approx (x \div_n z) \div_n y \mid n \geqslant 0\}$. Termination of $\overline{\mathcal{R}}/\mathcal{E}_{\mathrm{lab}}$ can be shown by the following polynomial interpretation: $[\mathsf{0}] = 0$, $[\mathsf{s}](x) = x + 1$, $x\,[-]\,y = x + y + 1$, and $x\,[\div_n]\,y = x + ny + n + y$ for all $n \geqslant 0$. According to Theorem 3 the original ETRS $\mathcal{R}/\mathcal{E}$ is terminating as well. Note that a direct termination proof with standard techniques is impossible since an instance of the last rule of $\mathcal{R}$ is self-embedding. In order to make this rule non-self-embedding it is essential that we label $\div$. This explains why Zantema's version of equational semantic labelling—presented in the next section—will fail here.

4 Semantic Labelling Cube

The original version of equational semantic labelling described in Zantema [24] is presented below.

Theorem 4 ([24]). *Let $\mathcal{R}/\mathcal{E}$ be an ETRS over a signature $\mathcal{F}$, $\mathcal{A}$ an $\mathcal{F}$-algebra, and ℓ a labelling for $\mathcal{A}$ such that function symbols occurring in $\mathcal{E}$ are unlabelled. If $\mathcal{A}$ is a model of $\mathcal{R}/\mathcal{E}$ then termination of $\mathcal{R}_{\mathrm{lab}}/\mathcal{E}$ is equivalent to termination of $\mathcal{R}/\mathcal{E}$.* □

In [24] it is remarked that the restriction that symbols in $\mathcal{E}$ are unlabelled is essential. Corollary 2, of which Theorem 4 is an immediate consequence, shows that this is not true. Zantema provides the non-terminating ETRS $\mathcal{R}/\mathcal{E}$ with $\mathcal{R} = \{(x + y) + z \to x + (y + z)\}$ and $\mathcal{E} = \{x + y \approx y + x\}$, and the model $\mathcal{A}$ consisting of the positive integers $\mathbb{N}_+$ with the function symbol $+$ interpreted as addition. By labelling $+$ with the value of its first argument, we obtain $\mathcal{R}_{\mathrm{lab}} = \{(x +_i y) +_{i+j} z \to x +_i (y +_j z) \mid i, j \in \mathbb{N}_+\}$ and $\mathcal{E}_{\mathrm{lab}} = \{x +_i y \approx y +_j x \mid i, j \in \mathbb{N}_+\}$. According to Corollary 2 the labelled ETRS $\mathcal{R}_{\mathrm{lab}}/\mathcal{E}_{\mathrm{lab}}$ is not terminating and indeed there are infinite rewrite sequences, e.g.

$$(x +_1 x) +_2 x \to x +_1 (x +_1 x) \sim (x +_1 x) +_2 x \to \cdots$$

In [24] it is remarked that $\mathcal{R}_{\mathrm{lab}}/\mathcal{E}'$ with $\mathcal{E}' = \{x +_i y \to y +_i x \mid i \in \mathbb{N}_+\}$ is terminating, since it is compatible with the polynomial interpretation in which the function symbol $+_i$ is interpreted as addition plus i, for every $i \in \mathbb{N}_+$. However, $\mathcal{E}'$ is *not* a labelled version of $\mathcal{E}$.

The various versions of equational semantic labelling presented above differ in three choices: (1) the order on the algebra $\mathcal{A}$ (partial order vs. quasi-order), (2) the relation between the algebra $\mathcal{A}$ and the ETRS $\mathcal{R}/\mathcal{E}$ (model vs. quasi-model), and (3) the labelling of the function symbols appearing in $\mathcal{E}$ (forbidden vs. allowed). This naturally gives rise to the cube of eight versions of equational semantic labelling possibilities shown in Figure 1. Every possibility is given as a string of three choices, each of them indicated by $-/+$ and ordered as above, so $-{+}{+}$ denotes the version of equational semantic labelling with partial order, quasi-model, and (possibly) labelled function symbols in $\mathcal{E}$. All eight versions of equational semantic labelling are sound, i.e., termination of the labelled ETRS

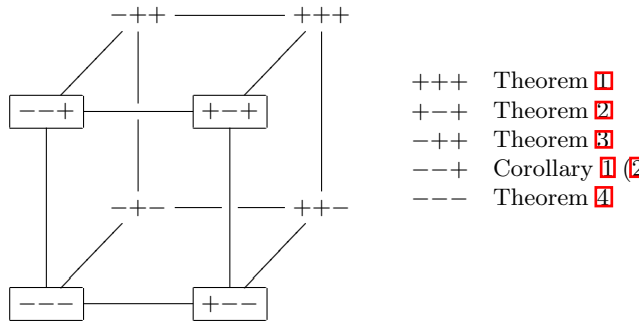


Fig. 1. Equational semantic labelling cube.

implies termination of the original ETRS. The versions in which termination of the labelled ETRS is equivalent to termination of the original ETRS are indicated by a surrounding box.

We present one more version of equational semantic labelling, stating that the implication of Theorem 1 becomes an equivalence in the special case that $\mathcal{E}$ is variable preserving (i.e., every equation $l \approx r \in \mathcal{E}$ has the property that l and r have the same number of occurrences of each variable), the (strict part of the) quasi-order $\succsim$ is well founded, and function symbols occurring in $\mathcal{E}$ are unlabelled. In other words, if $\mathcal{E}$ is variable preserving (which in particular is true for AC) and the quasi-order $\succsim$ is well founded then we can put a box around $++-$ in Figure 1. Before presenting the proof, we show the necessity of the three conditions. First consider the ETRS $\mathcal{R}/\mathcal{E}$ with $\mathcal{R} = \varnothing$ and $\mathcal{E} = \{\mathsf{f}(x, x) \approx x\}$ where the signature contains a unary function symbol g in addition to the function symbol f. Let $\mathcal{A}$ be the algebra over the carrier $\{0, 1\}$ with $1 \succ 0$ and operations $\mathsf{f}_{\mathcal{A}}(x, y) = x$ and $\mathsf{g}_{\mathcal{A}}(x) = x$. Note that $\mathcal{A}$ is a (quasi-)model of $\mathcal{R}/\mathcal{E}$. By labelling g with the value of its argument, we obtain the ETRS $\overline{\mathcal{R}}/\overline{\mathcal{E}}$ with $\overline{\mathcal{R}} = \mathrm{Dec} = \{\mathsf{g}_1(x) \to \mathsf{g}_0(x)\}$ and $\overline{\mathcal{E}} = \mathcal{E}$. The ETRS $\mathcal{R}/\mathcal{E}$ is trivially terminating, but $\overline{\mathcal{R}}/\overline{\mathcal{E}}$ admits the following infinite rewrite sequence:

$$\mathsf{g}_1(x) \sim \mathsf{f}(\mathsf{g}_1(x), \mathsf{g}_1(x)) \to \mathsf{f}(\mathsf{g}_0(x), \mathsf{g}_1(x)) \sim \mathsf{f}(\mathsf{g}_0(x), \mathsf{f}(\mathsf{g}_1(x), \mathsf{g}_1(x))) \to \cdots$$

Note that $\mathcal{E}$ is not variable preserving. The necessity of the well-foundedness of the quasi-order $\succsim$ follows by considering the terminating TRS $\mathcal{R}/\mathcal{E}$ with $\mathcal{R} = \{\mathsf{f}(x) \to \mathsf{g}(x)\}$ and $\mathcal{E} = \varnothing$, the algebra $\mathcal{A}$ over the carrier $\mathbb{Z}$ with standard order $\geqslant$ and operations $\mathsf{f}_{\mathcal{A}}(x) = \mathsf{g}_{\mathcal{A}}(x) = x$, and the labelling $\ell_{\mathsf{f}}(x) = x$. In this case we have $\mathcal{R}_{\mathrm{lab}} = \{\mathsf{f}_i(x) \to \mathsf{g}(x) \mid i \in \mathbb{Z}\}$ and $\mathrm{Dec} = \{\mathsf{f}_i(x) \to \mathsf{f}_j(x) \mid i > j\}$, so $\overline{\mathcal{R}}$ lacks termination. Finally, the requirement that function symbols occurring in $\mathcal{E}$ must be unlabelled is justified by the counterexample following Theorem 1.

Theorem 5. *Let $\mathcal{R}/\mathcal{E}$ be an ETRS over a signature $\mathcal{F}$ with $\mathcal{E}$ variable preserving, $(\mathcal{A}, \succsim)$ a monotone $\mathcal{F}$-algebra with $\succsim$ well-founded, and ℓ a monotone labelling for $(\mathcal{A}, \succsim)$ such that function symbols occurring in $\mathcal{E}$ are unlabelled. If $\mathcal{A}$ is a quasi-model of $\mathcal{R}/\mathcal{E}$ then termination of $\overline{\mathcal{R}}/\overline{\mathcal{E}}$ is equivalent to termination of $\mathcal{R}/\mathcal{E}$.*

Proof. First note that $\overline{\mathcal{R}}/\overline{\mathcal{E}} = (\mathcal{R}_{\mathrm{lab}} \cup \mathrm{Dec})/(\mathcal{E} \cup \mathrm{Eq})$ because function symbols occurring in $\mathcal{E}$ are unlabelled. The "if" part is a consequence of Theorem 1. For the "only if" part we show that the ETRS $\mathrm{Dec}/(\mathcal{E} \cup \mathrm{Eq})$ is terminating. For a term $t \in \mathcal{T}(\mathcal{F}_{\mathrm{lab}}, \mathcal{V})$ let $\phi(t)$ denote the multiset of all labels occurring in t. The following facts are not difficult to show:

- if $s \to_{\mathrm{Dec}} t$ then $\phi(s) \succ_{\mathrm{mul}} \phi(t)$,
- if $s \vdash\!\!\!\to_{\mathrm{Eq}} t$ then $\phi(s) \sim_{\mathrm{mul}} \phi(t)$,
- if $s \vdash\!\!\!\to_{\mathcal{E}} t$ then $\phi(s) = \phi(t)$.

Here $\succ_{\mathrm{mul}}$ denotes the multiset extension of $\succ$ ([5]) and $\sim_{\mathrm{mul}}$ denotes the multiset extension of the equivalence relation $\sim$ (which coincides with the equivalence relation associated with the multiset extension $\succsim_{\mathrm{mul}}$ of $\succsim$, see e.g. [17, Definition 5.6]). For the validity of the last observation it is essential that $\mathcal{E}$ is variable preserving and that function symbols occurring in $\mathcal{E}$ are unlabelled. From these facts and the well-foundedness of $\succsim_{\mathrm{mul}}$ we obtain the termination of $\mathrm{Dec}/(\mathcal{E} \cup \mathrm{Eq})$. Now, if $\overline{\mathcal{R}}/\overline{\mathcal{E}}$ is not terminating then it admits an infinite rewrite sequence which contains infinitely many $\mathcal{R}_{\mathrm{lab}}$-steps. Erasing all labels yields an infinite $\mathcal{R}/\mathcal{E}$-rewrite sequence, contradicting the assumption that $\mathcal{R}/\mathcal{E}$ is terminating. □

5 Dummy Elimination for Equational Rewriting

Ferreira, Kesner, and Puel [7] extended dummy elimination [8] to AC-rewriting by completely removing the AC-axioms. We show that their result is easily obtained in our equational semantic labelling framework. Our definition of $\mathrm{dummy}(\mathcal{R})$ is different from the one in [7,8], but easily seen to be equivalent.

Definition 3. *Let $\mathcal{R}$ be a TRS over a signature $\mathcal{F}$. Let e be a distinguished function symbol in $\mathcal{F}$ of arity $m \geqslant 1$ and let $\diamond$ be a fresh constant. We write $\mathcal{F}_\diamond$ for $(\mathcal{F} \setminus \{e\}) \cup \{\diamond\}$. The mapping* $\mathrm{cap}\colon \mathcal{T}(\mathcal{F}, \mathcal{V}) \to \mathcal{T}(\mathcal{F}_\diamond, \mathcal{V})$ *is inductively defined as follows:* $\mathrm{cap}(t) = t$ *if* $t \in \mathcal{V}$, $\mathrm{cap}(e(t_1, \ldots, t_m)) = \diamond$, *and* $\mathrm{cap}(f(t_1, \ldots, t_n)) = f(\mathrm{cap}(t_1), \ldots, \mathrm{cap}(t_n))$ *if* $f \neq e$. *The mapping* dummy *assigns to every term in $\mathcal{T}(\mathcal{F}, \mathcal{V})$ a subset of $\mathcal{T}(\mathcal{F}_\diamond, \mathcal{V})$:*

$$\mathrm{dummy}(t) = \{\mathrm{cap}(t)\} \cup \{\mathrm{cap}(s) \mid s \text{ is an argument of an } e \text{ symbol in } t\}.$$

Finally, we define

$$\mathrm{dummy}(\mathcal{R}) = \{\mathrm{cap}(l) \to r' \mid l \to r \in \mathcal{R} \text{ and } r' \in \mathrm{dummy}(r)\}.$$

Note that $\mathrm{dummy}(\mathcal{R})$ may contain invalid rewrite rules because $\mathrm{cap}(l)$ can have fewer variables than l. In that case, however, $\mathrm{dummy}(\mathcal{R})$ is not terminating and the results presented below hold vacuously. Ferreira and Zantema [8] showed that if $\mathrm{dummy}(\mathcal{R})$ is terminating then $\mathcal{R}$ is terminating. A simple proof of this fact using *self-labelling*, a special case of semantic labelling, can be found in Middeldorp *et al.* [16]. Two extensions of this result to equational rewriting are known. In [6] Ferreira showed that termination of $\mathcal{R}/\mathcal{E}$ follows from termination of $\mathrm{dummy}(\mathcal{R})/\mathcal{E}$ provided that $\mathcal{E}$ is variable preserving and does not contain the function symbol e. The extension presented in Ferreira *et al.* [7] is stated below.

Theorem 6. *Let $\mathcal{R}/\mathcal{E}$ be an ETRS with $\mathcal{E} = \mathrm{AC}(e)$. If* $\mathrm{dummy}(\mathcal{R})$ *is terminating then $\mathcal{R}/\mathcal{E}$ is terminating.*

In other words, AC-termination of $\mathcal{R}$ is reduced to termination of $\mathrm{dummy}(\mathcal{R})$.

Proof. We turn the set of terms $\mathcal{T}(\mathcal{F}_\diamond, \mathcal{V})$ into an $\mathcal{F}$-algebra $\mathcal{A}$ by defining $e_\mathcal{A}(t_1, \ldots, t_n) = \diamond$ and $f_\mathcal{A}(t_1, \ldots, t_n) = f(t_1, \ldots, t_n)$ for all other function symbols $f \in \mathcal{F}$ and terms $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F}_\diamond, \mathcal{V})$. We equip $\mathcal{A}$ with the (well-founded) partial order $\succeq = \to^*_{\mathrm{dummy}(\mathcal{R})}$. One can verify that $\mathcal{A}$ is monotone with respect to $\succeq$. An easy induction proof shows that $[\alpha](t) = \mathrm{cap}(t)\alpha$ for all terms $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$. We show that $\mathcal{A}$ is a quasi-model of $\mathcal{R}/\mathcal{E}$. Let $\alpha\colon \mathcal{V} \to \mathcal{T}(\mathcal{F}_\diamond, \mathcal{V})$ be an arbitrary assignment and let $l \to r \in \mathcal{R}$. We have $[\alpha](l) = \mathrm{cap}(l)\alpha$ and $[\alpha](r) = \mathrm{cap}(r)\alpha$ by the above property. The rewrite rule $\mathrm{cap}(l) \to \mathrm{cap}(r)$ belongs to $\mathrm{dummy}(\mathcal{R})$ by definition and hence $[\alpha](l) \succeq [\alpha](r)$. For the two equations $l \approx r \in \mathcal{E}$ we clearly have $[\alpha](l) = \diamond = [\alpha](r)$. Hence $\mathcal{A}$ is a quasi-model of $\mathcal{R}/\mathcal{E}$.

Define the (monotone) labelling ℓ as follows: $\ell_f = f_\mathcal{A}$ for all function symbols $f \in \mathcal{F}$. According to Theorem 3 it is sufficient to show that $\overline{\mathcal{R}}/\mathcal{E}_{\mathrm{lab}}$ is terminating. Define a precedence $\sqsupset$ on $\mathcal{F}_{\mathrm{lab}}$ as follows: $f_s \sqsupset g_t$ if and only if $s\ (\succ \cup \rhd)^+\ t$, where $\rhd$ is the proper superterm relation. Note that $\sqsupset$ inherits well-foundedness from $\succ$. We claim that $\overline{\mathcal{R}}$ is precedence terminating with respect to $\sqsupset$. Rewrite rules in Dec are of the form $f_s(x_1, \ldots, x_n) \to f_t(x_1, \ldots, x_n)$ with $s \succ t$ and thus $f_s \sqsupset f_t$. For rules in $\mathcal{R}_{\mathrm{lab}}$ we make use of the following property:

$$\text{if } t \trianglelefteq r \text{ then } \mathrm{cap}(t) \trianglelefteq r' \text{ for some term } r' \in \mathrm{dummy}(r). \qquad (*)$$

Now let $l \to r \in \mathcal{R}_{\mathrm{lab}}$. By definition there exist an assignment $\alpha\colon \mathcal{V} \to \mathcal{T}(\mathcal{F}_\diamond, \mathcal{V})$ and a rewrite rule $l' \to r' \in \mathcal{R}$ such that $l = \mathrm{lab}_\alpha(l')$ and $r = \mathrm{lab}_\alpha(r')$. The label of the root symbol of l is $[\alpha](l') = \mathrm{cap}(l')\alpha$. Let s be the label of a function symbol in r. By construction $s = [\alpha](t) = \mathrm{cap}(t)\alpha$ for some subterm t of r'. According to (1) we have $\mathrm{cap}(t) \trianglelefteq r''$ for some $r'' \in \mathrm{dummy}(r')$. By definition $\mathrm{cap}(l') \to r'' \in \mathrm{dummy}(\mathcal{R})$ and hence $\mathrm{cap}(l')\alpha \succ r''\alpha \trianglerighteq \mathrm{cap}(t)\alpha = s$. Consequently, $\mathrm{root}(l) \sqsupset f$ for every function symbol f in r. This completes the proof of precedence termination of $\overline{\mathcal{R}}$. Since $\mathcal{E}_{\mathrm{lab}} = \mathrm{AC}(e_\diamond)$, termination of $\overline{\mathcal{R}}/\mathcal{E}_{\mathrm{lab}}$ follows from Lemma 1. □

The reader is invited to compare our proof with the one in [7]. For the above simple proof we indeed needed our new powerful version of equational semantic labelling, i.e., Zantema's restricted version (Theorem 4) would not have worked.

One may wonder whether the soundness proof of the version of equational dummy elimination presented in [6] can also be simplified by equational semantic labelling. This turns out not to be the case. One reason is that function symbols of $\mathcal{E}$ that also appear in $\mathcal{R}$ will be labelled, causing $\mathcal{E}_{\mathrm{lab}}$ (and $\overline{\mathcal{E}}$) to be essentially different from $\mathcal{E}$. In particular, if $\mathcal{E}$ consists of AC-axioms then $\mathcal{E}_{\mathrm{lab}}$ contains non-AC axioms and hence AC-compatible orders are not applicable to $\overline{\mathcal{R}}/\overline{\mathcal{E}}$. Moreover, Lemma 1 does not extend to arbitrary ESs $\mathcal{E}$ and it is unclear how to change the definition of precedence termination such that it does.

Recently, Nakamura and Toyama [18] improved dummy elimination by restricting r' in the definition of $\mathrm{dummy}(\mathcal{R})$ to terms in $(\mathrm{dummy}(r) \setminus \mathcal{T}(\mathcal{F}_\mathcal{C}, \mathcal{V})) \cup$

$\{\mathrm{cap}(r)\}$ with $\mathcal{F}_\mathcal{C}$ denoting the constructors of $\mathcal{R}$. In other words, elements of $\mathrm{dummy}(r) \setminus \{\mathrm{cap}(r)\}$ that do not contain a defined function symbol need not be considered when forming the right-hand sides of the rewrite rules in $\mathrm{dummy}(\mathcal{R})$. For example, the TRS $\mathcal{R} = \{\mathsf{f}(\mathsf{a}) \to \mathsf{f}(\mathsf{b}), \mathsf{b} \to \mathsf{e}(\mathsf{a})\}$ is transformed into the non-terminating TRS $\mathrm{dummy}(\mathcal{R}) = \{\mathsf{f}(\mathsf{a}) \to \mathsf{f}(\mathsf{b}), \mathsf{b} \to \diamond, \mathsf{b} \to \mathsf{a}\}$ by dummy elimination whereas the above improvement yields the terminating TRS $\{\mathsf{f}(\mathsf{a}) \to \mathsf{f}(\mathsf{b}), \mathsf{b} \to \diamond\}$. Aoto[1] suggested that a further improvement is possible by stripping off the outermost constructor context of every element in $\mathrm{dummy}(r) \setminus \{\mathrm{cap}(r)\}$. For $\mathcal{R} = \{\mathsf{f}(\mathsf{a}(x)) \to \mathsf{f}(\mathsf{b}), \mathsf{b} \to \mathsf{e}(\mathsf{a}(\mathsf{f}(\mathsf{c})))\}$ this would yield the terminating TRS $\{\mathsf{f}(\mathsf{a}(x)) \to \mathsf{f}(\mathsf{b}), \mathsf{b} \to \diamond, \mathsf{b} \to \mathsf{f}(\mathsf{c})\}$ whereas the transformation of [18] produces $\mathrm{dummy}(\mathcal{R}) = \{\mathsf{f}(\mathsf{a}(x)) \to \mathsf{f}(\mathsf{b}), \mathsf{b} \to \diamond, \mathsf{b} \to \mathsf{a}(\mathsf{f}(\mathsf{c}))\}$, which is clearly not terminating.

These ideas are easily incorporated in our definition of dummy elimination. Here $\mathcal{F}_\mathcal{D} = \mathcal{F} \setminus \mathcal{F}_\mathcal{C}$ denotes the defined symbols of $\mathcal{R}$.

Definition 4. *Let $\mathcal{R}$ be a TRS over a signature $\mathcal{F}$. The mapping* dummy' *assigns to every term in $\mathcal{T}(\mathcal{F}, \mathcal{V})$ a subset of $\mathcal{T}(\mathcal{F}_\diamond, \mathcal{V})$, as follows:*

$$\mathrm{dummy}'(t) = \mathrm{cap}(t) \cup \left\{ \mathrm{cap}(s) \;\middle|\; \begin{array}{l} s \text{ is a maximal subterm of an argument} \\ \text{of } e \text{ in } t \text{ such that } \mathrm{root}(s) \in \mathcal{F}_\mathcal{D} \setminus \{e\} \end{array} \right\}.$$

We define

$$\mathrm{dummy}'(\mathcal{R}) = \{\mathrm{cap}(l) \to r' \mid l \to r \in \mathcal{R} \text{ and } r' \in \mathrm{dummy}'(r)\}.$$

Theorem 7. *Let $\mathcal{R}/\mathcal{E}$ be an ETRS with $\mathcal{E} = \mathrm{AC}(e)$. If* $\mathrm{dummy}'(\mathcal{R})$ *is terminating then $\mathcal{R}/\mathcal{E}$ is terminating.*

Proof. Very similar to the proof of Theorem 6. The difference is that we do not label the function symbols in $\mathcal{F}_\mathcal{C}$. In order to obtain precedence termination of $\overline{\mathcal{R}}$ we extend the precedence $\sqsupset$ on $\mathcal{F}_{\mathrm{lab}}$ by $f_t \sqsupset g$ for every $f \in \mathcal{F}_\mathcal{D}$, $t \in \mathcal{T}(\mathcal{F}_\diamond, \mathcal{V})$, and $g \in \mathcal{F}_\mathcal{C}$. In addition, $(*)$ is replaced by the following property:

$$\text{if } t \trianglelefteq r \text{ and } \mathrm{root}(t) \in \mathcal{F}_\mathcal{D} \text{ then } \mathrm{cap}(t) \trianglelefteq r' \text{ for some term } r' \in \mathrm{dummy}'(r).$$

Taking these changes into consideration, termination of $\mathcal{R}/\mathcal{E}$ is obtained as in the proof of Theorem 6. □

6 Distribution Elimination for Equational Rewriting

Next we show that our results on equational semantic labelling can also be used to extend the distribution elimination transformation of [23] to the AC case. Again, for that purpose we need our powerful version of equational semantic labelling, i.e., Theorem 4 does not suffice. Let $\mathcal{R}$ be a TRS over a signature $\mathcal{F}$

[1] Remark made at the 14th Japanese Term Rewriting Meeting, Nara Institute of Science and Technology, March 15–16, 1999.

and let $e \in \mathcal{F}$ be a designated function symbol whose arity is at least one. A rewrite rule $l \to r \in \mathcal{R}$ is called a *distribution rule* for e if $l = C[e(x_1, \ldots, x_m)]$ and $r = e(C[x_1], \ldots, C[x_m])$ for some non-empty context C in which e does not occur and pairwise different variables $x_1, \ldots, x_m$. Distribution elimination is a technique that transforms $\mathcal{R}$ by eliminating all distribution rules for e and removing the symbol e from the right-hand sides of the other rules. Let $\mathcal{F}_{\mathrm{distr}} = \mathcal{F} \setminus \{e\}$. We inductively define a mapping distr that assigns to every term in $\mathcal{T}(\mathcal{F}, \mathcal{V})$ a non-empty subset of $\mathcal{T}(\mathcal{F}_{\mathrm{distr}}, \mathcal{V})$, as follows:

$$\mathrm{distr}(t) = \begin{cases} \{t\} & \text{if } t \in \mathcal{V}, \\ \bigcup_{i=1}^{m} \mathrm{distr}(t_i) & \text{if } t = e(t_1, \ldots, t_m), \\ \{f(s_1, \ldots, s_n) \mid s_i \in \mathrm{distr}(t_i)\} & \text{if } t = f(t_1, \ldots, t_n) \text{ with } f \neq e. \end{cases}$$

It is extended to rewrite systems as follows:

$$\mathrm{distr}(\mathcal{R}) = \{l \to r' \mid l \to r \in \mathcal{R} \text{ is no distribution rule for } e \text{ and } r' \in \mathrm{distr}(r)\}.$$

A rewrite system is called *right-linear* if no right-hand side of a rule contains multiple occurrences of the same variable. The following theorem extends Zantema's soundness result for distribution elimination to the AC case.

Theorem 8. *Let $\mathcal{R}/\mathcal{E}$ be an ETRS with $\mathcal{E} = \mathrm{AC}(e)$ such that e does not occur in the left-hand sides of rewrite rules of $\mathcal{R}$ that are not distribution rules for e. If* $\mathrm{distr}(\mathcal{R})$ *is terminating and right-linear then $\mathcal{R}/\mathcal{E}$ is terminating.*

Proof. We turn the set of finite non-empty multisets over $\mathcal{T}(\mathcal{F}_{\mathrm{distr}}, \mathcal{V})$ into an $\mathcal{F}$-algebra $\mathcal{A}$ by defining

$$f_{\mathcal{A}}(M_1, \ldots, M_n) = \begin{cases} \{f(t_1, \ldots, t_n) \mid t_i \in M_i \text{ for all } 1 \leqslant i \leqslant n\} & \text{if } f \neq e, \\ M_1 \cup M_2 & \text{if } f = e \end{cases}$$

for all function symbols $f \in \mathcal{F}$ and finite non-empty multisets $M_1, \ldots, M_n$ of terms in $\mathcal{T}(\mathcal{F}_{\mathrm{distr}}, \mathcal{V})$. (Note that $n = 2$ if $f = e$.) We equip $\mathcal{A}$ with the (well-founded) partial order $\succcurlyeq = \succ^{=}_{\mathrm{mul}}$ where $\succ = \to^{+}_{\mathrm{distr}(\mathcal{R})}$. One easily shows that $(\mathcal{A}, \succcurlyeq)$ is a monotone $\mathcal{F}$-algebra. It can be shown (cf. the nontrivial proof of Theorem 12 in [23]) that

1. $l =_{\mathcal{A}} r$ for every distribution rule $l \to r \in \mathcal{R}$,
2. $l \succcurlyeq_{\mathcal{A}} r$ for every other rule $l \to r \in \mathcal{R}$.

For (2) we need the right-linearity assumption of $\mathrm{distr}(\mathcal{R})$. From the definition of $e_{\mathcal{A}}$ we obtain $e(x, y) =_{\mathcal{A}} e(y, x)$ and $e(e(x, y), z) =_{\mathcal{A}} e(x, e(y, z))$. Hence $(\mathcal{A}, \succcurlyeq)$ is a quasi-model of $\mathcal{R}/\mathcal{E}$.

Define the (monotone) labelling ℓ as follows: $\ell_f = f_{\mathcal{A}}$ for all function symbols $f \neq e$. According to Theorem 3 it is sufficient to show that $\overline{\mathcal{R}}/\mathcal{E}_{\mathrm{lab}}$ is terminating. Define the precedence $\sqsupset$ on $\mathcal{F}_{\mathrm{lab}}$ as follows: $f \sqsupset g$ if and only if either $f \neq e$ and $g = e$ or $f = f'_M$ and $g = g'_N$ with $M\ ((\succ \cup \rhd)^+)_{\mathrm{mul}}\ N$. Note that $\sqsupset$ is well founded. We claim that $\overline{\mathcal{R}}$ is precedence terminating with respect to $\sqsupset$. Rewrite

rules in Dec are of the form $f_M(x_1,\dots,x_n) \to f_N(x_1,\dots,x_n)$ with $M \succ_{\mathrm{mul}} N$ and thus $f_M \sqsupset f_N$. For rules in $\mathcal{R}_{\mathrm{lab}}$ we make use of the following property, which is not difficult to prove:

3. if $t \lhd r$ then $[\alpha](r) \rhd_{\mathrm{mul}} [\alpha](t)$ for every assignment α.

Now let $l \to r \in \mathcal{R}_{\mathrm{lab}}$. By definition there is an assignment $\alpha\colon \mathcal{V} \to \mathcal{T}(\mathcal{F}_{\mathrm{distr}}, \mathcal{V})$ and a rewrite rule $l' \to r' \in \mathcal{R}$ such that $l = \mathrm{lab}_\alpha(l')$ and $r = \mathrm{lab}_\alpha(r')$. Since $\mathrm{root}(l') \neq e$, the label of the root symbol of l is $[\alpha](l')$. If e occurs in r' then $\mathrm{root}(l) \sqsupset e$ by definition. Let M be the label of a function symbol in r. By construction $M = [\alpha](t)$ for some subterm t of r'. We distinguish two cases. First consider the case that $l' \to r' \in \mathcal{R}$ is a distribution rule. Because $\mathrm{root}(r') = e$, t is a proper subterm of r'. Property (3) yields $[\alpha](r') \rhd_{\mathrm{mul}} [\alpha](t)$. We have $[\alpha](l') = [\alpha](r')$ by (1). Hence $[\alpha](l')\ ((\succ \cup \rhd)^+)_{\mathrm{mul}}\ M$ as required. Next let $l' \to r' \in \mathcal{R}$ be a non-distribution rule. From (3) we infer that $[\alpha](r') \trianglerighteq_{\mathrm{mul}} [\alpha](t)$ (if $t = r'$ then $[\alpha](r') = [\alpha](t)$ holds). According to (2) we have $[\alpha](l') \succ_{\mathrm{mul}} [\alpha](r')$. Hence also in this case we obtain $[\alpha](l')\ ((\succ \cup \rhd)^+)_{\mathrm{mul}}\ M$. This completes the proof of precedence termination of $\overline{\mathcal{R}}$. Since $\mathcal{E}_{\mathrm{lab}} = \mathcal{E} = \mathrm{AC}(e)$, termination of $\overline{\mathcal{R}}/\mathcal{E}_{\mathrm{lab}}$ follows from Lemma 1. □

Next we show that the right-linearity requirement in the preceding theorem can be dropped if termination is strengthened to *total termination*. A TRS is called totally terminating if it is compatible with a well-founded monotone algebra in which the underlying order is total. Since adding a constant to the signature does not affect total termination, from now on we assume that the set of ground terms is non-empty. Total termination is equivalent (see [9, Theorem 13]) to compatibility with a well-founded monotone total order on ground terms. Here, "compatibility" means that $l\sigma \succ r\sigma$ holds for all rules $l \to r \in \mathcal{R}$ and all substitutions such that $l\sigma$ is a ground term. It should be noted that standard termination techniques like polynomial interpretations, recursive path order, and Knuth-Bendix order all yield total termination.

Theorem 9. *Let $\mathcal{R}/\mathcal{E}$ be an ETRS with $\mathcal{E} = \mathrm{AC}(e)$ such that e does not occur in the left-hand sides of rewrite rules of $\mathcal{R}$ that are not distribution rules for e. If* $\mathrm{distr}(\mathcal{R})$ *is totally terminating then $\mathcal{R}/\mathcal{E}$ is terminating.*

Proof. There is a well-founded monotone total order $\succ$ on $\mathcal{T}(\mathcal{F}_{\mathrm{distr}})$ which is compatible with $\mathrm{distr}(\mathcal{R})$. We turn $\mathcal{T}(\mathcal{F}_{\mathrm{distr}})$ into an $\mathcal{F}$-algebra $\mathcal{A}$ by defining $f_{\mathcal{A}}(t_1,\dots,t_n) = f(t_1,\dots,t_n)$ if $f \neq e$ and $f_{\mathcal{A}}(t_1,\dots,t_n) = \max\{t_1,t_2\}$ if $f = e$ for all symbols $f \in \mathcal{F}$ and terms $t_1,\dots,t_n$ in $\mathcal{T}(\mathcal{F}_{\mathrm{distr}})$. We equip $\mathcal{A}$ with the (well-founded) partial order $\succeq$. One can show that $(\mathcal{A}, \succeq)$ is a monotone $\mathcal{F}$-algebra. It is not difficult to verify that $l =_{\mathcal{A}} r$ for every distribution rule $l \to r \in \mathcal{R}$ and the two equations $l \approx r \in \mathcal{E}$. An easy induction proof shows that

1. for all terms $r \in \mathcal{T}(\mathcal{F},\mathcal{V})$ and assignments α there exists a term $s \in \mathrm{distr}(r)$ such that $[\alpha](r) = [\alpha](s)$.

Using this property, we obtain (by induction on r) that $l \succ_{\mathcal{A}} r$ for every non-distribution rule $l \to r \in \mathcal{R}$. Hence $(\mathcal{A}, \succeq)$ is a quasi-model of $\mathcal{R}/\mathcal{E}$.

Define the (monotone) labelling ℓ as follows: $\ell_f = f_{\mathcal{A}}$ for all function symbols $f \neq e$. According to Theorem 3 it is sufficient to show that $\overline{\mathcal{R}}/\mathcal{E}_{\mathrm{lab}}$ is terminating. Define the precedence $\sqsupset$ on $\mathcal{F}_{\mathrm{lab}}$ as follows: $f \sqsupset g$ if and only if either $f \neq e$ and $g = e$ or $f = f'_s$ and $g = g'_t$ with $s\ (\succ \cup \rhd)^+\ t$. Note that $\sqsupset$ is well founded. The following property is not difficult to prove:

2. if $t \lhd r$ then $[\alpha](r) \trianglerighteq [\alpha](t)$ for every assignment α.

However, $[\alpha](r) \rhd [\alpha](t)$ need not hold (consider e.g. $t \lhd e(t,t)$) and as a consequence the labelled distribution rules in $\overline{\mathcal{R}}$ are not precedence terminating with respect to $\sqsupset$. Nevertheless, the precedence termination of the labelled non-distribution rules in $\mathcal{R}_{\mathrm{lab}}$ as well as the rules in Dec is obtained as in the proof of Theorem 8. Hence any AC-compatible recursive path order $\sqsupset^{\mathrm{AC}}_{\mathrm{rpo}}$ induced by the precedence $\sqsupset$ that is defined on terms with variables (cf. the proof of Lemma 1) will orient these rules from left to right. Let $l = C[e(x,y)] \to e(C[x], C[y]) = r$ be a distribution rule in $\mathcal{R}$ and let α be an arbitrary assignment. We claim that $\mathrm{lab}_\alpha(l) \sqsupset^{\mathrm{AC}}_{\mathrm{rpo}} \mathrm{lab}_\alpha(r)$. Since $C \neq \Box$, $\mathrm{root}(\mathrm{lab}_\alpha(l)) \sqsupset e = \mathrm{root}(\mathrm{lab}_\alpha(r))$ by definition. It suffices to show that $\mathrm{lab}_\alpha(l) \sqsupset^{\mathrm{AC}}_{\mathrm{rpo}} \mathrm{lab}_\alpha(C[x])$ and $\mathrm{lab}_\alpha(l) \sqsupset^{\mathrm{AC}}_{\mathrm{rpo}} \mathrm{lab}_\alpha(C[y])$. We have $\mathrm{lab}_\alpha(C[x]) = C_1[x]$, $\mathrm{lab}_\alpha(C[y]) = C_2[y]$ for some labelled contexts C_1 and C_2, and $\mathrm{lab}_\alpha(l) = C_1[e(x,y)]$ if $\alpha(x) \succeq \alpha(y)$ and $\mathrm{lab}_\alpha(l) = C_2[e(x,y)]$ otherwise. We consider only the case $\alpha(x) \succeq \alpha(y)$ here. We have $C_1[e(x,y)] \sqsupset^{\mathrm{AC}}_{\mathrm{rpo}} C_1[x]$ by the subterm property of $\sqsupset^{\mathrm{AC}}_{\mathrm{rpo}}$. If $\alpha(x) = \alpha(y)$ then $C_2[y] = C_1[y]$ and thus also $C_1[e(x,y)] \sqsupset^{\mathrm{AC}}_{\mathrm{rpo}} C_2[y]$ by the subterm property. If $\alpha(x) \succ \alpha(y)$ then $C_1[e(x,y)] \sqsupset^{\mathrm{AC}}_{\mathrm{rpo}} C_2[y]$ because the rewrite rule $C_1[e(x,y)] \to C_2[y]$ is precedence terminating. This can be seen as follows. The label of the root symbol of $C_1[e(x,y)]$ is $[\alpha](C[x])$. Let q be the label of a function symbol in $C_2[y]$. By construction $q = [\alpha](t)$ for some subterm t of $C[y]$. We obtain $[\alpha](C[y]) \trianglerighteq [\alpha](t) = q$ from (2). The monotonicity of $\mathcal{A}$ yields $[\alpha](C[x]) \succ [\alpha](C[y])$. Hence $[\alpha](C[x])\ (\succ \cup \rhd)^+\ q$ as desired. We conclude that $\overline{\mathcal{R}}/\mathcal{E}_{\mathrm{lab}}$ is terminating. Theorem 3 yields the termination of $\mathcal{R}/\mathcal{E}$. □

The above theorem extends a similar result for TRSs in Zantema [23]. Actually, in [23] it is shown that $\mathcal{R}$ is *totally* terminating if $\mathrm{distr}(\mathcal{R})$ is totally terminating. Our semantic labelling proof does not give total termination of $\mathcal{R}/\mathcal{E}$. Nevertheless, the more complicated proof in [23] can be extended to deal with $\mathrm{AC}(e)$, so $\mathcal{R}/\mathcal{E}$ is in fact totally terminating.

In Middeldorp *et al.* [16] it is shown that for $\mathcal{E} = \varnothing$ the right-linearity requirement in Theorem 8 can be dropped if there are no distribution rules in $\mathcal{R}$. It remains to be seen whether this result is also true if $\mathcal{E} = \mathrm{AC}(e)$. We note that the semantic labelling proof in [16] does not extend to $\mathcal{R}/\mathcal{E}$ because the interpretation of e defined there, an arbitrary projection function, is inconsistent with the commutativity of e.

Acknowledgements. We are grateful to the anonymous referees for their careful reading. Aart Middeldorp is partially supported by the Grant-in-Aid for Scientific Research C(2) 11680338 of the Ministry of Education, Science, Sports and Culture of Japan. Jürgen Giesl is supported by the DFG under grant GI 274/4-1.

References

1. T. Arts and J. Giesl, *Termination of Term Rewriting Using Dependency Pairs*, Theoretical Computer Science 236, pp. 133–178, 2000.
2. A. Ben Cherifa and P. Lescanne, *Termination of Rewriting Systems by Polynomial Interpretations and its Implementation*, Science of Computer Programming 9(2), pp. 137–159, 1987.
3. F. Baader and T. Nipkow, *Term Rewriting and All That*, Cambridge University Press, 1998.
4. N. Dershowitz, *Termination of Rewriting*, Journal of Symbolic Computation 3, pp. 69–116, 1987.
5. N. Dershowitz and Z. Manna, *Proving Termination with Multiset Orderings*, Communications of the ACM 22(8), pp. 465–476, 1979.
6. M.C.F. Ferreira, *Dummy Elimination in Equational Rewriting*, Proc. 7th RTA, LNCS 1103, pp. 78–92, 1996.
7. M.C.F. Ferreira, D. Kesner, and L. Puel, *Reducing AC-Termination to Termination*, Proc. 23rd MFCS, LNCS 1450, pp. 239–247, 1998.
8. M.C.F. Ferreira and H. Zantema, *Dummy Elimination: Making Termination Easier*, Proc. 10th FCT, LNCS 965, pp. 243–252, 1995.
9. M.C.F. Ferreira and H. Zantema, *Total Termination of Term Rewriting*, Applicable Algebra in Engineering, Communication and Computing 7, pp. 133–162, 1996.
10. A. Geser, *Relative Termination*, Ph.D. thesis, Universität Passau, 1990.
11. J.-P. Jouannaud and M. Muñoz, *Termination of a Set of Rules Modulo a Set of Equations*, Proc. 7th CADE, LNCS 170, pp. 175–193, 1984.
12. D. Kapur and G. Sivakumar, *A Total, Ground Path Ordering for Proving Termination of AC-Rewrite Systems*, Proc. 8th RTA, LNCS 1232, pp. 142–155, 1997.
13. D. Kapur, G. Sivakumar, and H. Zhang, *A New Method for Proving Termination of AC-Rewrite Systems*, Proc. 10th FSTTCS, LNCS 472, pp. 133–148, 1990.
14. C. Marché and X. Urbain, *Termination of Associative-Commutative Rewriting by Dependency Pairs*, Proc. 9th RTA, LNCS 1379, pp. 241–255, 1998.
15. A. Middeldorp and H. Ohsaki, *Type Introduction for Equational Rewriting*, Acta Informatica, 2000. To appear.
16. A. Middeldorp, H. Ohsaki, and H. Zantema, *Transforming Termination by Self-Labelling*, Proc. 13th CADE, LNAI 1104, pp. 373–387, 1996.
17. A. Middeldorp, H. Zantema, *Simple Termination of Rewrite Systems*, Theoretical Computer Science 175, pp. 127–158, 1997.
18. M. Nakamura and Y. Toyama, *On Proving Termination by General Dummy Elimination*, Technical report of IEICE, COMP 98-58 (1998-11), pp. 57–64, 1998. In Japanese.
19. A. Rubio, *A Fully Syntactic AC-RPO*, Proc. 10th RTA, LNCS 1631, pp. 133–147, 1999.
20. A. Rubio and R. Nieuwenhuis, *A Total AC-Compatible Ordering Based on RPO*, Theoretical Computer Science 142, pp. 209–227, 1995.
21. J. Steinbach, *Termination of Rewriting: Extensions, Comparison and Automatic Generation of Simplification Orderings*, Ph.D. thesis, Univ. Kaiserslautern, 1994.
22. Y. Toyama, *Counterexamples to Termination for the Direct Sum of Term Rewriting Systems*, Information Processing Letters 25, pp. 141–143, 1987.
23. H. Zantema, *Termination of Term Rewriting: Interpretation and Type Elimination*, Journal of Symbolic Computation 17, pp. 23–50, 1994.
24. H. Zantema, *Termination of Term Rewriting by Semantic Labelling*, Fundamenta Informaticae 24, pp. 89–105, 1995.

On the Computational Interpretation of Negation

Michel Parigot

Equipe de Logique Mathématique
case 7012, Université Paris 7
2 place Jussieu, 75251 Paris cedex 05, France

Abstract. We investigate the possibility of giving a computational interpretation of an involutive negation in classical natural deduction. We first show why this cannot be simply achieved by adding $\neg\neg A = A$ to typed λ-calculus: the main obstacle is that an involutive negation cannot be a particular case of implication at the computational level. It means that one has to go out typed λ-calculus in order to have a safe computational interpretation of an involutive negation.
We then show how to equip $\lambda\mu$-calculus in a natural way with an involutive negation: the abstraction and application associated to negation are simply the operators μ and $[\,]$ from $\lambda\mu$-calculus. The resulting system is called symmetric $\lambda\mu$-calculus.
Finally we give a translation of symmetric λ-calculus in symmetric $\lambda\mu$-calculus, which doesn't make use of the rule of μ-reduction of $\lambda\mu$-calculus (which is precisely the rule which makes the difference between classical and intuitionistic proofs in the context of $\lambda\mu$-calculus). This seems to indicate that an involutive negation generates an original way of computing. Because symmetric $\lambda\mu$-calculus contains both ways, it should be a good framework for further investigations.

1 Introduction

A lot of efforts have been done in the past 10 years to give computational interpretations of classical logic, starting from the work of Felleisen [5,6], Griffin [9] and Murthy [15]. It has been shown that classical natural deduction allows to modelize imperative features added to functional languages like Scheme, Common Lisp or ML. Two particular systems, λ_C-calculus ([5], [6]) and $\lambda\mu$-calculus ([17]), have been intensively studied and the relation between features of languages, rules of natural deduction, machines and semantics seems to be well understood.

In the context of sequent calculus, several other computational interpretations of classical logic have been constructed following the spirit of Girard's linear logic [7]. It is often claimed in this context that computational interpretations of negation in classical logic should be involutive, that is $\neg\neg A = A$ should be

P. Clote and H. Schwichtenberg (Eds.): CSL 2000, LNCS 1862, pp. 472–484, 2000.

realised at the computational level. It is even sometimes claimed that this is the distinguishing feature of classical logic. But the real computational effect of the involutive character is not clear.

Systems coming from a natural deduction setting, like λ_C-calculus or $\lambda\mu$-calculus, don't have an involutive negation. There is only one exception: the symmetric λ-calculus of Barbanera and Berardi [2,3], which is explicitely based on an involutive negation, but whose concrete programming counterpart is not so well understood.

This paper is devoted to the study of the possibility of having an involutive negation in a computational interpretation of the usual natural deduction system.

In section 2 we discuss in details the possiblity of adding $\neg\neg A = A$ to typed λ-calculus (as a way of adding the classical absurdity rule to intuitionistic natural deduction). We show that there are two obstacles: negation cannot be a particular case of implication and $\bot$ cannot be an atomic type, contrary to the use coming from intuitionistic logic. The fact that negation and implication need to have different computational interpretations means that one has to go out typed λ-calculus in order to have a safe computational interpretation of an involutive negation.

In section 3 we show how to equip $\lambda\mu$-calculus in a natural way with an involutive negation: the abstraction and application associated to negation are simply the operators μ and $[\,]$ from $\lambda\mu$-calculus. The resulting system is called symmetric $\lambda\mu$-calculus.

In section 4 we give a translation of symmetric λ-calculus in symmetric $\lambda\mu$-calculus, which doesn't make use of the rule of μ-reduction of $\lambda\mu$-calculus (which is precisely the rule which makes the difference between classical and intuitionistic proofs in the context of $\lambda\mu$-calculus). This seems to indicate that an involutive negation generates an original way of computing. Because symmetric $\lambda\mu$-calculus contains both ways, it should be a good framework for further investigations.

In the sequel types are designated by letters A, B, C etc., while atomic types are designated by P, Q, R, etc. Terms of λ-calculus are constructed upon variables x, y, z using two rules:

(abstraction) if x is a variable and u a term, then $\lambda x.u$ is a term.

(application) if u and v are terms, then $(u)v$ is a term.

Reduction of λ-calculus is denoted by $\triangleright$.

2 About Typed λ-Calculus and $\neg\neg A = A$

Let us consider usual typed λ-calculus whose types are constructed from atomic types using $\to$, $\neg$ and $\bot$ ($\bot$ is considered as an atomic type). We denote this system by $S_{\to,\neg,\bot}$. Judgements are expressions of the form $\Gamma \vdash u : A$, where A is a type, u is a term of λ-calculus and Γ is a context of the form $x_1 : A_1, ..., x_n : A_n$.

The rules of derivation of $S_{\to,\neg,\bot}$ are the following:

$$x : A \vdash x : A$$

$$\frac{\Gamma,\ x : A \vdash u : B}{\Gamma \vdash \lambda x.u : A \to B} \qquad \frac{\Gamma_1 \vdash u : A \to B \quad \Gamma_2 \vdash v : A}{\Gamma_1, \Gamma_2 \vdash (u)v : B}$$

$$\frac{\Gamma,\ x : A \vdash u : \bot}{\Gamma \vdash \lambda x.u : \neg A} \qquad \frac{\Gamma_1 \vdash u : \neg A \quad \Gamma_2 \vdash v : A}{\Gamma_1, \Gamma_2 \vdash (u)v : \bot}$$

We adopt in these rules an implicit management of contraction and weakening. Contraction is obtained through the fact that contexts are considered as sets: in a conclusion of a rule a context Γ_1, Γ_2 denotes the union of the contexts Γ_1 and Γ_2. Weakening is obtained by the convention that in a premise of an introduction rules, $\Gamma,\ x : A$ denotes a context where $x : A$ doesn't necessary appear.
Note that in $S_{\to,\neg,\bot}$, $\neg A$ is identified with $A \to \bot$. Indeed $\neg$ is often considered as a derived connective whose definition is precisely $\neg A = A \to \bot$.
Suppose now that we add the rule $\neg\neg A \subseteq A$, i.e.

$$\frac{\Gamma \vdash u\ :\ \neg\neg A}{\Gamma \vdash u\ :\ A}$$

which is equivalent (up to η-equivalence) to the trivial interpretation of the absurdity rule

$$\frac{\Gamma, x\ :\ \neg A \vdash u\ : \bot}{\Gamma \vdash \lambda x.u\ :\ A}$$

We call the resulting system $S^*_{\to,\neg,\bot}$. In this system one can prove $x : A \vdash \lambda k.(k)x : A$ as follows

$$\frac{\dfrac{k : \neg A \vdash k : \neg A \qquad x : A \vdash x : A}{k : \neg A, x : A \vdash (k)x : \bot}}{x : A \vdash \lambda k.(k)x : A}$$

The term $\lambda k.(k)x$ will play a fundamental role in the examples of sections 2.1 and 2.2.
We show in the next sections that the system $S^*_{\to,\neg,\bot}$ doesn't satisfy normalisation and correctness properties. This means that the addition of the rule $\neg\neg A \subseteq A$ (and a fortiori the addition of $\neg\neg A = A$) to $S_{\to,\neg,\bot}$ destroys normalisation and correctness properties.

2.1 Normalisation

Proposition 1. *Let $\theta = \lambda f.\lambda x.(\lambda k.(k)f)(f)x$. The term $((\theta)\theta)\theta$ is typable in $S^*_{\to,\neg,\bot}$ and not normalisable.*

Proof. Let C be a type. One defines C^n by induction on n by: $C^1 = C$ and $C^{n+1} = C^n \to C^n$.
One proves that $\vdash \theta : C^{n+2}$, for each $n \geq 1$. We have

$$f : C^{n+1}, x : C^n \vdash (f)x : C^n$$

Because $f : C^{n+1} \vdash \lambda k.(k)f : C^{n+1}$, we have also

$$f : C^{n+1}, x : C^n \vdash (\lambda k.(k)f)(f)x : C^n$$

and thus $\vdash \lambda f.\lambda x.(\lambda k.(k)f)(f)x : C^{n+2}$, i.e $\vdash \theta : C^{n+2}$.
It follows that $\vdash ((\theta)\theta)\theta : C^{n+2}$, for each $n \geq 1$: it suffices to type the first occurence of θ with C^{n+4}, the second with C^{n+3} and the third with C^{n+2}.
Now it is easy to check that $((\theta)\theta)\theta$ is not normalisable because
$\theta = \lambda f.\lambda x.(\lambda k.(k)f)(f)x$ and θ reduces in one step to $\theta^1 = \lambda f.\lambda x.((f)x)f$ and $((\theta^1)\theta^1)\theta^1$ has only one reduction sequence and reduces to itself in two steps as follows:

$$\begin{aligned}((\theta^1)\theta^1)\theta^1 = &\ ((\lambda f.\lambda x.((f)x)f)\theta^1)\theta^1 \\ \triangleright &\ (\lambda x.((\theta^1)x)\theta^1)\theta^1 \\ \triangleright &\ ((\theta^1)\theta^1)\theta^1\end{aligned}$$

2.2 Correctness

In $S^*_{\rightarrow,\neg,\perp}$ types are not preserved by reduction in an essential way, which forbids the derivation of correct programs from proofs. This loss of correctness can be easily shown if one extends typed λ-calculus to a second order typed λ-calculus. Let us take for example the simplest such system, due to Leivant [14] and widely developed in [12,13], which allows to derive correct programs from equational specifications of functions. In such a system one can easely prove that $\lambda x.\lambda y.(x)y$ is a program which computes the exponential y^x. More precisely one has a term e, $\beta\eta$-equivalent to $\lambda x.\lambda y.(x)y$ such that:

$\vdash e : \forall u \forall v(Nu \rightarrow (Nv \rightarrow Nv^u))$

where Nx is the second order type $\forall X(\forall y(Xy \rightarrow Xsy) \rightarrow (X0 \rightarrow Xx))$ saying that x is a natural number.
If one adds $\neg\neg A \subseteq A$, one can prove that $\lambda x.\lambda y.(y)x$ is a also a program which computes the exponential y^x. In other words, the calculus mixed up x^y and y^x! This forbids obviously any hope to derive correct programs in this calculus.

Proof. Suppose $\vdash e : \forall u \forall v(Nu \rightarrow (Nv \rightarrow Nv^u))$. Then
$x : Nu \vdash (e)x : Nv \rightarrow Nv^u$ and $y : \neg(Nv \rightarrow Nv^u), x : Nu \vdash (y)(e)x : \perp$. It follows $x : Nu \vdash \lambda y.(y)(e)x : \neg\neg(Nv \rightarrow Nv^u)$ and because $\neg\neg A \subseteq A$,
$x : Nu \vdash \lambda y.(y)(e)x : Nv \rightarrow Nv^u$.
Therefore $\vdash \lambda x.\lambda y.(y)(e)x : Nu \rightarrow Nv \rightarrow Nv^u$ and
$\vdash \lambda x.\lambda y.(y)(e)x : \forall u \forall v(Nu \rightarrow (Nv \rightarrow Nv^u))$.
This means that $\lambda x.\lambda y.(y)(e)x$ is also a program for y^x. But $\lambda x.\lambda y.(y)(e)x$ is $\beta\eta$-equivalent to $\lambda x.\lambda y.(y)x$:

$$\begin{aligned}\lambda x.\lambda y.(y)(e)x \equiv_{\beta\eta} &\ \lambda x.\lambda y.(y)(\lambda x.\lambda y.(x)y)x \\ \equiv_{\beta\eta} &\ \lambda x.\lambda y.(y)\lambda y.(x)y \\ \equiv_{\beta\eta} &\ \lambda x.\lambda y.(y)x\end{aligned}$$

2.3 Discussion

The problem behind the examples of sections 2.1 and 2.2 appears clearly in the following derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{k : \neg(A \to B) \vdash k : \neg(A \to B) \quad f : A \to B \vdash f : A \to B}{k : \neg(A \to B), f : A \to B \vdash (k)f : \bot}}{f : A \to B \vdash \lambda k.(k)f : \neg\neg(A \to B)}}{f : A \to B \vdash \lambda k.(k)f : A \to B} \qquad x : A \vdash x : A}{f : A \to B, x : A \vdash (\lambda k.(k)f)x : B}$$

This derivation shows that in $S^*_{\to,\neg,\bot}$, the term $(\lambda k.(k)f)x$ is typable of type B in the context $f : A \to B, x : A$. But $(\lambda k.(k)f)x$ reduces to the term $(x)f$, which is not typable in the context $f : A \to B, x : A$. Therefore typing in $S^*_{\to,\neg,\bot}$ is not preserved under reduction.
This derivation also shows that the addition of the trivial absurdity rule to typed λ-calculus produces the effect of adding the following rule:

$$\dfrac{\Gamma_1 \vdash u : A \to B \quad \Gamma_2 \vdash v : A}{\Gamma_1, \Gamma_2 \vdash (v)u : B}$$

to the usual rule of elimination of implication:

$$\dfrac{\Gamma_1 \vdash u : A \to B \quad \Gamma_2 \vdash v : A}{\Gamma_1, \Gamma_2 \vdash (u)v : B}$$

The effect of choosing an involutive negation is indeed to induce a symetry at the level of application. As the application associated to $\to$ cannot be symmetric, the only possibility to get a safe calculus with an involutive negation is to keep separated the computational interpretations of $\neg$ and $\to$. The obvious way of doing is to choose two different abstractions and two different applications.
As shown below there is one more obstacle to an involutive negation in the context of typed λ-calculus.

2.4 The Role of $\bot$

Suppose now that we restrict our system by forgetting $\to$. The resulting system $S^*_{\neg,\bot}$ has the following rules:

$$x : A \vdash x : A$$

$$\dfrac{\Gamma\, x : A \vdash u : \bot}{\Gamma \vdash \lambda x.u : \neg A} \qquad \dfrac{\Gamma_1 \vdash u : \neg A \quad \Gamma_2 \vdash v : A}{\Gamma_1, \Gamma_2 \vdash (u)v : \bot}$$

and in addition the trivial absurdity rule:

$$\dfrac{\Gamma, x \,:\, \neg A \vdash u \,:\, \bot}{\Gamma \vdash \lambda x.u \,:\, A}$$

We show that normalisation fails for $S^*_{\neg,\bot}$.

Proposition 2. *Let* $\xi = \lambda x.\lambda f.(\lambda k.(k)f)(f)x$ *and* y *a variable. The term* $(\lambda k.(k)\xi)(\xi)y$ *is typable in* $S^*_{\neg,\bot}$ *and not normalisable.*

Proof. We first show that $(\lambda k.(k)\xi)(\xi)y$ is typable in $S^*_{\neg,\bot}$.

We have $x : \bot, f : \neg\bot \vdash (f)x : \bot$ and $f : \neg\bot \vdash \lambda k.(k)f : \neg\bot$. Therefore $x : \bot\,, f : \neg\bot \vdash (\lambda k.(k)f)(f)x : \bot$ and $x : \bot \vdash \lambda f.(\lambda k.(k)f)(f)x : \neg\neg\bot$. Because $\neg\neg\bot \subseteq \bot$, we also $\vdash \lambda x.\lambda f.(\lambda k.(k)f)(f)x : \neg\bot$ i.e. $\vdash \xi : \neg\bot$. It follows $\vdash \lambda k.(k)\xi : \neg\bot$ and $y : \bot \vdash (\lambda k.(k)\xi)(\xi)y : \bot$.

Let $\xi' = \lambda x.\lambda f.((f)x)f$. The term $(\lambda k.(k)\xi)(\xi)y$ reduces to the term $((\xi')y)\xi'$ which has only one reduction sequence and reduces in two steps to itself as follows:

$$\begin{aligned} ((\xi')y)\xi' = {} & ((\lambda x.\lambda f.((f)x)f)y)\xi' \\ \triangleright {} & (\lambda f.((f)y)f)\xi' \\ \triangleright {} & ((\xi')y)\xi' \end{aligned}$$

In order to type a non normalisable term in the system $S^*_{\neg,\bot}$ we have made an essential use of the fact that $\bot$ is an atomic type of the system, which can be used to built other type (we used the type $\neg\bot$). The problem lies in the confusion between two uses of $\bot$: as indicating a contradiction in a proof and as an atomic type. Therefore in order to get normalising calculus with an involutive negation we have to forbid $\bot$ as an atomic type (it can be a "special" type, which is outside the system).

Note that the two obstacles to an involutive computational interpretation of negation are completely different. In particular, the examples of sections 2.1 and 2.2 do not use the fact that $\bot$ is an atomic type of the system: they hold for the system $S^*_{\to,\neg}$, where $\bot$ is used only for indicating a contradiction in a proof.

3 Typed $\lambda\mu$-Calculus with $\neg\neg A = A$

In this section, we extend typed $\lambda\mu$-calculus in a natural way with an involutive negation: the abstraction and application associated to negation are simply the operator μ and $[\,]$ from $\lambda\mu$-calculus.

3.1 $\lambda\mu$-Calculus

Typed $\lambda\mu$-calculus is a simple computational interpretation of classical logic introduced in [17]. It has both a clear interpretation in terms of environment machines and a clear semantics in terms of continuations [10,11,20].

The $\lambda\mu$-calculus has two kinds of variables: the λ-variables $x, y, z, ...$, and the μ-variables $\alpha, \beta, \gamma, ...$ Terms are defined inductively as follows:

- x is a term, for x a λ-variable;
- $\lambda x.u$ is a term, for x a λ-variable and u a term;
- $(t)u$ is a term, for t and u terms;
- $\mu\alpha.[\beta]t$ is a term, for t a term and α, β μ-variables.

Expressions of the form $[\beta]t$, where β is μ-variables and t a term, are called named terms. They correspond to type $\bot$ in typed $\lambda\mu$-calculus.

Typed $\lambda\mu$-calculus is a calculus for classical logic, enjoying confluence and strong normalisation [17,18], which doesn't make use of negation. Types are build from atomic types using $\rightarrow$ only. Type $\bot$ is not needed, but is added for convenience as a special type denoting a contradiction in a proof (in the context of typed $\lambda\mu$-calculus one could also consider it as an atomic type). Judgments have two contexts: one to the left for λ-variables and one to the right for μ-variables. In order to make the symmetric extension easier to understand, we adopt here a presentation where the right context is replaced by a negated left context. Of course, this doesn't change the calculus; in particular negation is not needed inside types.
Judgments are expressions of the form $\Gamma;\ \Delta\ \vdash\ u : A$, where A is a type, u is a term of $\lambda\mu$-calculus, Γ is a context of the form $x_1 : A_1, ..., x_n : A_n$ and Δ a context of the form $\alpha_1 : \neg A_1, ..., \alpha_n : \neg A_n$.
The typing rules of $\lambda\mu$-calculus are the following:

$$x : A\ \vdash\ x : A$$

$$\frac{\Gamma,\ x : A;\ \Delta\ \vdash\ u : B}{\Gamma;\ \Delta\ \vdash\ \lambda x.u : A \rightarrow B} \qquad \frac{\Gamma_1;\ \Delta_1\ \vdash\ u : A \rightarrow B \quad \Gamma_2;\ \Delta_2\ \vdash\ v : A}{\Gamma_1, \Gamma_2;\ \Delta_1, \Delta_2\ \vdash\ (u)v : B}$$

$$\frac{\Gamma;\ \Delta,\ \alpha : \neg A\ \vdash\ u : \bot}{\Gamma;\ \Delta\ \vdash\ \mu\alpha.u : A} \qquad \frac{\Gamma;\ \Delta\ \vdash\ u : A}{\Gamma;\ \Delta,\ \alpha : \neg A\ \vdash\ [\alpha]u : \bot}$$

As for typed λ-calculus we adopt in these rules an implicit management of contraction and weakening, with the same conventions as in section 2.
The $\lambda\mu$-calculus has two fundamental reduction rules:

(R_1) $\quad (\lambda x.u)v\ \triangleright\ u[v/x]$

(R_2) $\quad (\mu\alpha.u)v\ \triangleright\ \mu\alpha'.u[[\alpha'](w)v/[\alpha]w]$

and in addition simplification rules (like η-rule of λ-calculus):

(S_1) $\quad \lambda x.(u)x\ \triangleright\ u$

(S_2) $\quad \mu\alpha.[\alpha]u\ \triangleright\ u$

Simplification rules are subject to the following restrictions: in (S_1), x has no free occurences in u; in (S_2), α has no free occurences in u.
In (R_2), the term $u[[\alpha'](w)v/[\alpha]w]$ is defined as the result of substituting to each subterm of u of the form $[\alpha]w$, the new subterm $[\alpha'](w)v$. Note that if α has type $\neg(A \rightarrow B)$, then α' has type $\neg B$.

3.2 Symmetric $\lambda\mu$-Calculus

We introduce an extension of $\lambda\mu$-calculus with an involutive negation, called symmetric $\lambda\mu$-calculus. We simply take the abstraction μ and the application $[\,]$ of $\lambda\mu$-calculus as beeing the new abstraction and application corresponding to the computational content of this involutive negation.

Types of symmetric $\lambda\mu$-calculus are defined as follows:

$$A := P \mid \neg P \mid A \to B \mid \neg(A \to B)$$

where P denotes atomic types.
The negation $\neg$ is extended to an involutive negation on types in the obvious way.
For convenience, one adds a special type $\bot$. For the reason explained in section 2.4 , $\bot$ doesn't belong to the set of atomic types.
Note that an involutive negation, invites to confuse the rule of introduction of negation

$$\frac{\Gamma,\ x : A \ \vdash\ u : \bot}{\Gamma \ \vdash\ \mu x.u : \neg A}$$

and the absurdity rule

$$\frac{\Gamma,\ x : \neg A \ \vdash\ u : \bot}{\Gamma \ \vdash\ \mu x.u : A}$$

and also to have only one kind of variable.
This is this drastic solution that we adopt with symmetric $\lambda\mu$-calculus, because it should better capture the essence of an involutive negation, but more permissive ones might also be interesting at the computational level.

Terms of Symmetric $\lambda\mu$-Calculus.

Symmetric $\lambda\mu$-calculus has only one kind of variables. Terms are defined inductively as follows:

- x is a term, for x a λ-variable;
- $\lambda x.u$ is a term, for x a variable and u a term;
- $(t)u$ is a term, for t and u terms;
- $\mu x.[u]v$ is a term, for x a variable and u, v terms.

Expressions of the form $[u]v$, where u, v are terms, are called named terms. They correspond to type $\bot$ in typed symmetric $\lambda\mu$-calculus.

Typing Rules of Symmetric $\lambda\mu$-Calculus.

$$x : A \ \vdash\ x : A$$

$$\frac{\Gamma,\ x : A \ \vdash\ u : B}{\Gamma \ \vdash\ \lambda x.u : A \to B} \qquad \frac{\Gamma_1 \ \vdash\ u : A \to B \quad \Gamma_2 \ \vdash\ v : A}{\Gamma_1, \Gamma_2 \ \vdash\ (u)v : B}$$

$$\frac{\Gamma, x : A \ \vdash\ u : \bot}{\Gamma \ \vdash\ \mu x.u : \neg A} \qquad \frac{\Gamma_1 \ \vdash\ u : \neg A \quad \Gamma_2 \ \vdash\ v : A}{\Gamma_1, \Gamma_2 \ \vdash\ [u]v : \bot}$$

As for typed λ-calculus one adopts in these rules an implicit management of contraction and weakening, with the same conventions as in section 2.

Reduction Rules of Symmetric $\lambda\mu$-calculus.

The symmetric $\lambda\mu$-calculus has the following reduction rules:

(R_1) $(\lambda x.u)v \triangleright u[v/x]$

(R_2) $(\mu x.u)v \triangleright \mu x'.u[\mu z.[x'](z)v/x]$

(R_3) $[\mu x.u]v \triangleright u[v/x]$

(R_4) $[u]\mu x.v \triangleright v[u/x]$

and in addition simplification rules (like η-rule of λ-calculus):

(S_1) $\lambda x.(u)x \triangleright u$

(S_2) $\mu x.[x]u \triangleright u$

(S_3) $\mu x.[u]x \triangleright u$

Simplification rules are subject to the following restriction: in (S_1), (S_2), (S_3), x has no free occurences in u.

Symmetric $\lambda\mu$-calculus is clearly an extension of $\lambda\mu$-calculus. Rule (R1) is the usual β-reduction of λ-calculus. Because we make a more liberal use of variables in symmetric $\lambda\mu$-calculus, the rule (R2) of μ-reduction is stated in a more general setting than the corresponding rule of $\lambda\mu$-calculus, but his effect is exactly the same when restricted to terms of $\lambda\mu$-calculus.

In $\lambda\mu$-calculus the substituted variable x (which is a μ-variable) always occurs in a subterm $[x]w$ and the result of the substitution is in this case $[\mu z.[x'](z)v]w$ which reduces to $[x'](w)v$. This corresponds exactly to the substitution of the rule (R2) of $\lambda\mu$-calculus.

The reduction rule (R2) of symmetric $\lambda\mu$-calculus is simpler to understand with typed terms. Suppose that one reduces the term $(\mu x^{\neg(A\to B)}.u^{\perp})v^A$. One replaces in u the occurences of $x^{\neg(A\to B)}$ by a canonical term of type $\neg(A \to B)$ which is $\mu z^{A\to B}.[x'^{\neg B}](z^{A\to B})v^A$. This term can be thought as a pair $\langle v^A, x'^{\neg B}\rangle$.

The two new rules are the rules (R3) and (R4), which correspond respectively to a kind β-reduction for μ and its symmetric, are exactly those of the symmetric λ-calculus of Barbanera and Berardi [2].

Note that the rule (R2) introduces a "communication" between $\to$ and $\neg$, which has no real equivalent in the symmetric λ-calculus.

Because of its symmetric nature (appearing in rules (R3) and (R4)), symmetric $\lambda\mu$-calculus is essentially not confluent. As in symmetric λ-calculus, this non confluence could be used in positive way to derive symmetric programs.

Indeed symmetric $\lambda\mu$-calculus contains two different ways of computing with classical proofs: the one of $\lambda\mu$-calculus, based on the specific rule (R2) of μ-reduction, which is well understood in terms of machines and continuations; the one of symmetric λ-calculus, based on rules (R3) and (R4), which is of a different computational nature. The embedding of $\lambda\mu$-calculus in symmetric $\lambda\mu$-calculus is obvious and doesn't make use of (R3) and (R4). In the next section we develop

an embedding of symmetric λ-calculus in symmetric $\lambda\mu$-calculus, which doesn't make use of (R2).

4 Interpretation of Symmetric λ-Calculus in Symmetric $\lambda\mu$-Calculus

In this section, we give a translation of symmetric λ-calculus in symmetric $\lambda\mu$-calculus, which doesn't make use of rule (R2): reduction involves only the rules (R1), (R3) and (R4).

4.1 The Symmetric λ-Calculus of Barbanera and Berardi

The types of the system are defined by:

$$A := P|\neg P|A \wedge B|A \vee B$$

where P denotes atomic types.
An involutive negation on types is defined as follows:

$\neg(A) = \neg A$
$\neg(\neg A) = A$
$\neg(A \wedge B) = \neg A \vee \neg B$
$\neg(A \vee B) = \neg A \wedge \neg B$

There is also a special type $\perp$, which doesn't belong to the set of atomic types.

Derivation Rules of Symmetric λ-Calculus

$x : A \vdash x : A$

$$\frac{\Gamma \vdash u : A \qquad \Delta \vdash v : B}{\Gamma, \Delta \vdash \langle u, v\rangle : A \wedge B}\ \wedge\text{-}intro \qquad \frac{\Gamma \vdash u_i : A_i}{\Gamma \vdash \sigma_i(u_i) : A_1 \vee A_2}\ \vee\text{-}intro\ (i{=}1{,}2)$$

$$\frac{\Gamma, x : A \vdash u : \perp}{\Gamma \vdash \lambda x.u : \neg A}\ \neg\text{-}intro \qquad \frac{\Gamma \vdash u : \neg A \qquad \Delta \vdash v : A}{\Gamma, \Delta \vdash u * v : \perp}\ \neg\text{-}elim$$

As for typed λ-calculus one adopts in these rules an implicit management of contraction and weakening, with the same conventions as in section 2.

Reduction Rules of Symmetric λ-Calculus

(β) $\quad \lambda x.u * v \ \triangleright\ u[v/x]$
$(\beta^{\perp})$ $\quad u * \lambda x.v \ \triangleright\ v[u/x]$
(π) $\quad \langle u_1, u_2\rangle * \sigma_i(v_i) \ \triangleright\ u_i * v_i$
$(\pi^{\perp})$ $\quad \sigma_i(v_i) * \langle u_1, u_2\rangle \ \triangleright\ v_i * u_i$

Symmetric λ-calculus is obviously not confluent but enjoys strong normalisation [2,3]. Moreover its non-confluence can be used in a positive way to derive symmetric programs [4].

4.2 Interpretation of the Symmetric λ-Calculus in the Symmetric $\lambda\mu$-Calculus

Connectives $\wedge$ and $\vee$ are translated as follows:

$A \vee B = \neg A \to B$

$A \wedge B = \neg(A \to \neg B)$

The rules $\wedge$-intro, $\vee$-intro, $\neg$-intro and $\neg$-elim are translated as follows:

$\wedge$-intro

$$\dfrac{\dfrac{\dfrac{z : A \to \neg B \vdash z : A \to \neg B \qquad \Gamma_1 \vdash u : A}{\Gamma_1, z : A \to \neg B \vdash (z)u : \neg B} \qquad \Gamma_2 \vdash v : B}{\Gamma_1, \Gamma_2, z : A \to \neg B \vdash [(z)u]v : \bot}}{\Gamma_1, \Gamma_2 \vdash \mu z.[(z)u]v : \neg(A \to \neg B)}$$

$\vee$-intro$_1$

$$\dfrac{\dfrac{\dfrac{z : \neg A \vdash z : \neg A \qquad \Gamma \vdash u : A}{\Gamma, z : \neg A \vdash [z]u : \bot}}{\Gamma, z : \neg A \vdash \mu d.[z]u : B}}{\Gamma \vdash \lambda z.\mu d.[z]u : \neg A \to B}$$

$\vee$-intro$_2$

$$\dfrac{\dfrac{\dfrac{z : \neg B \vdash z : \neg B \qquad \Gamma \vdash u : B}{\Gamma, z : \neg B \vdash [z]u : \bot}}{\Gamma \vdash \mu z.[z]u : B}}{\Gamma \vdash \lambda d.\mu z.[z]u : \neg A \to B}$$

$\neg$-intro

$$\dfrac{\Gamma, x : A \vdash u : \bot}{\Gamma \vdash \mu x.u : \neg A}$$

$\neg$-elim

$$\dfrac{\Gamma_1 \vdash u : \neg A \qquad \Gamma_2 \vdash v : A}{\Gamma_1, \Gamma_2 \vdash [u]v : \bot}$$

Let T be the translation defined inductively as follows:

$T(x) = x$

$T(\langle u, v\rangle) = \mu z[(z)T(u)]T(v)$

$T(\sigma_1(u) = \lambda z.\mu d.[z]T(u)$

$T(\sigma_2(u) = \lambda d.\mu z.[z]T(u)$

$T(\lambda x.u) = \mu x.T(u)$

$T(u * v) = [T(u)]T(v)$

As shown before, T preserves types and it is easy to check that T is compatible with substitution, i.e. $T(u[v/x]) = T(u)[T(v)/x]$. We prove now that T preserves reduction, in a way which doesn't make use of rule (R2).

$$\begin{aligned}
T(\{\lambda x.u\} * v) &= [\mu x.T(u)]T(v) \\
&\triangleright\ T(u)[T(v)/x] \\
&= T(u[v/x]) \\
T(\langle u_1, u_2\rangle *\ \sigma_1(v)) &= [\mu z.[(z)T(u_1)]T(u_2)]\lambda z.\mu d.[z]T(v) \\
&\triangleright\ [(\lambda z.\mu d.[z]T(v))T(u_1)]T(u_2) \\
&\triangleright\ [\mu d.[T(u_1)]T(v)]T(u_2) \\
&\triangleright\ [T(u_1)]T(v) \\
&= T(u_1 * v) \\
T(\langle u_1, u_2\rangle *\ \sigma_2(v)) &= [\mu z.[(z)T(u_1)]T(u_2)]\lambda d.\mu z.[z]T(v) \\
&\triangleright\ [(\lambda d.\mu z.[z]T(v))T(u_1)]T(u_2) \\
&\triangleright\ [\mu z.[z]T(v)]T(u_2) \\
&\triangleright\ [T(u_2)]T(v) \\
&= T(u_2 * v)
\end{aligned}$$

The symmetric rules are preserved in the same way.

References

1. H. Barendregt : The Lambda-Calculus. North-Holland, 1981.
2. F. Barbanera, S. Berardi : A symmetric lambda-calculus for classical program extraction. Proceedings TACS'94, Springer LNCS **789** (1994).
3. F. Barbanera, S. Berardi : A symmetric lambda-calculus for classical program extraction. Information and Computation **125** (1996) 103-117.
4. F. Barbanera, S. Berardi, M. Schivalocchi : "Classical" programming-with-proofs in lambda-sym: an analysis of a non-confluence. Proc. TACS'97.
5. M. Felleisen, D.P. Friedman, E. Kohlbecker, B. Duba : A syntactic theory of sequential control. Theoretical Computer Science **52** (1987) pp 205-237.
6. M. Felleisen, R. Hieb : The revised report on the syntactic theory of sequential control and state. Theoretical Computer Science **102** (1994) 235-271.
7. J.Y. Girard : Linear logic. Theoretical Computer Science. **50** (1987) 1-102.
8. J.Y. Girard, Y. Lafont, and P. Taylor : Proofs and Types. Cambridge University Press, 1989.
9. T. Griffin : A formulae-as-types notion of control. Proc. POPL'90 (1990) 47-58.
10. M. Hofmann, T. Streicher : Continuation models are universal for $\lambda\mu$-calculus. Proc. LICS'97 (1997) 387-397.
11. M. Hofmann, T. Streicher : Completeness of continuation models for $\lambda\mu$-calculus. Information and Computation (to appear).
12. J.L. Krivine, M. Parigot: Programming with proofs. J. of Information Processing and Cybernetics **26** (1990) 149-168.
13. J.L. Krivine : Lambda-calcul, types et modèles. Masson, 1990.
14. D. Leivant : Reasoning about functional programs and complexity classes associated with type disciplines. Proc. FOCS'83 (1983) 460-469.
15. C. Murthy : Extracting Constructive Content from Classical Proofs. PhD Thesis, Cornell, 1990.

16. M. Parigot : Free Deduction: an Analysis of "Computations" in Classical Logic. Proc. Russian Conference on Logic Programming, 1991, Springer LNCS **592** 361-380.
17. M. Parigot : $\lambda\mu$-calculus: an Algorithmic Interpretation of Classical Natural Deduction. Proc. LPAR'92, Springer LNCS **624** (1992) 190-201.
18. M. Parigot : Strong normalisation for second order classical natural deduction, Proc. LICS'93 (1993) 39-46.
19. C.H.L. Ong, C.A. Stewart : A Curry-Howard foundation for functional computation with control. Proc. POPL'97 (1997)
20. P. Selinger : Control categories and duality: on the categorical semantics of lambda-mu calculus, Mathematical Structures in Computer Science (to appear).

From Programs to Games: Invariance and Safety for Bisimulation

Marc Pauly

Center for Mathematics and Computer Science (CWI)
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
pauly@cwi.nl

Abstract. Bisimulation is generalized from process models to game models which are described using *Game Logic* (*GL*), a logic which extends Propositional Dynamic Logic by an additional operator *dual* which allows for the construction of complex 2-player games. It is shown that bisimilar states satisfy the same *GL*-formulas (invariance), and that an atomic bisimulation can be lifted to non-atomic *GL*-games (safety). Over process models, *GL* forms a highly expressive fragment of the modal μ-calculus, and within first-order logic, the game operations of *GL* are complete: they suffice to construct all first-order definable games which are monotonic and safe for bisimulation.

1 Introduction

Among the different notions of process equivalence one can consider, bisimulation has received much attention especially within the logic community. From the perspective of modal logic, there is a tight correspondence between bisimilar states of a process (Kripke model) and states which make the same modal formulas true: Bisimilar states satisfy the same modal formulas, and for certain classes of Kripke models (e.g. finite models), the converse holds as well. This bisimulation-invariance result makes bisimulation an attractive notion of equivalence between Kripke models, since it matches the expressive power of the modal language rather well. On the other hand, bisimulation has provided a characterization of the modal fragment of first-order logic (*FOL*). Modal formulas can be translated into formulas of *FOL*, and it turns out (see [5] and lemma 2) that the *modal fragment* of *FOL* is precisely its bisimulation-invariant fragment.

This line of investigation and the two main results mentioned can be extended from modal logic to Propositional Dynamic Logic (*PDL*) [12,16], a logic where the modalities are indexed by programs. Programs can be constructed from atomic programs using a number of program operations such as sequential composition, iteration, etc., and like modal formulas, *PDL*-formulas are bisimulation-invariant. Secondly, iteration-free *PDL*-programs can be translated into *FOL* as well, raising the question how to characterize the *FOL*-fragment which (translations of) *PDL*-programs define. In [6], such a result has been obtained: The *program-fragment* of *FOL* can be characterized as its bisimulation-safe fragment,

P. Clote and H. Schwichtenberg (Eds.): CSL 2000, LNCS 1862, pp. 485–496, 2000.

where roughly speaking a program is safe for bisimulation if it preserves bisimulation. This result shows that if we take bisimulation as our notion of process equivalence and *FOL* as our language, the program operations provided by *PDL* are complete, i.e. no additional program operations will allow us to construct new programs. This result has been extended to monadic second-order logic in [13].

In this paper, we carry the investigation one step further, moving from nondeterministic programs (i.e. 1-player games) to 2-player games. In the program specification literature, such a move has been useful to obtain intermediate nonimplementable specifications which contain demonic as well as angelic choices [2,3]. A formalism such as the refinement calculus models programs and specifications as predicate transformers, i.e. functions which map postconditions to weakest preconditions. This notion is general enough to model games as well as programs, and it is the semantic foundation of Game Logic (*GL*), introduced in [18]. In *GL*, the program operations of *PDL* are extended with a new construct called *dual*. In the terminology of games, this operation introduces a role switch between the players.

After introducing game models and *GL* in the next section, section 3 introduces bisimulation for game models. The first main result of this paper (proposition 1) shows that *GL*-formulas are invariant and *GL*-operations safe for bisimulation. Starting from section 4, we focus on a special class of models, Kripke models. For Kripke models, the generalized notion of bisimulation coincides with standard bisimulation and *GL* becomes a fragment of the modal μ-calculus which can express properties requiring multiple nested fixpoints. Section 5 is devoted to the second main result (proposition 2): Over Kripke models, iteration-free games (like programs) can be translated into *FOL*, thus defining the *game-fragment* of *FOL*. The result demonstrates that this fragment is precisely the monotonic bisimulation-safe fragment of *FOL*.

2 Syntax and Semantics of Game Logic

GL is a logic to reason about winning strategies in strictly competitive determined games between two players who we shall call Angel and Demon. For a game expression γ, the formula $\langle\gamma\rangle\varphi$ will express that Angel has a strategy in game γ for achieving φ, i.e. he can guarantee that the terminal position reached after γ has been played satisfies φ. Similarly, $[\gamma]\varphi$ will express that Demon has a strategy in game γ for achieving φ.

GL provides a number of operations which allow for the construction of complex games: A test game $\varphi?$ consists of checking through a neutral arbiter whether proposition φ holds at that state. If it does, nothing happens (i.e. another game can be played) and otherwise, Demon wins. The game $\gamma_1 \cup \gamma_2$ gives Angel the choice of playing γ_1 or γ_2. The sequential composition $\gamma_1;\gamma_2$ of two games consists of first playing γ_1 and then γ_2, and in the iterated game γ^*, Angel can choose how often to play γ, possibly not at all. More precisely, after each

play of γ, Angel can decide whether or not to play γ another time, but γ may not be played infinitely often (in that case, Demon wins).

In order to introduce interaction between the players, *GL* adds an operator *dual* for role interchange: Playing the dual game γ^{d} is the same as playing γ with the roles of the players reversed, i.e. any choice made by Angel in γ will be made by Demon in γ^{d} and vice versa.

Formally, the language of *GL* consists of two sorts, games and propositions. Given a set of atomic games Γ_0 and a set of atomic propositions Φ_0, games γ and propositions φ can have the following syntactic forms, yielding the set of games Γ and the set of propositions/formulas Φ:

$$\begin{aligned}\gamma &:= g \mid \varphi? \mid \gamma;\gamma \mid \gamma \cup \gamma \mid \gamma^* \mid \gamma^{\mathsf{d}}\\ \varphi &:= \bot \mid p \mid \neg\varphi \mid \varphi \vee \varphi \mid \langle\gamma\rangle\varphi\end{aligned}$$

where $p \in \Phi_0$ and $g \in \Gamma_0$. As usual, we define $\top := \neg\bot$, $[\gamma]\varphi := \neg\langle\gamma\rangle\neg\varphi$, $\varphi \wedge \psi := \neg(\neg\varphi \vee \neg\psi)$, $\varphi \to \psi := \neg\varphi \vee \psi$ and $\varphi \leftrightarrow \psi := (\varphi \to \psi) \wedge (\psi \to \varphi)$.

As for the semantics, given a signature (Φ_0, Γ_0) of atomic propositions and atomic games, a game model (also called neighborhood model or minimal model, see [9]) $\mathcal{I} = (S, \{N_g \mid g \in \Gamma_0\}, \{V_p \mid p \in \Phi_0\})$, consists of a set of states S, a valuation for each propositional letter $p \in \Phi_0$ such that $V_p \subseteq S$, and a function $N_g : \mathcal{P}(S) \to \mathcal{P}(S)$ for every atomic game $g \in \Gamma_0$. We require monotonicity, i.e. $X \subseteq Y$ implies $N_g(X) \subseteq N_g(Y)$ for all $g \in \Gamma_0$.

Intuitively, we can think of every state s as being associated with a 2-player game tree for every atomic game $g \in \Gamma_0$. Every terminal position of such a game tree is associated with a state $t \in S$. Since generally both players will have choices in the game, a player will usually not be able to force a particular state to come about at the end of the game. Rather, all he can do is force the outcome to lie in a particular set $Y \subseteq S$, and the game model specifies the sets of states which Angel can force, i.e. $s \in N_g(Y)$ holds if Angel has a strategy for ending up at a terminal position whose associated state is in Y. Given this interpretation, the monotonicity requirement is a natural one: If Angel has a strategy to bring about a state in Y, then that strategy trivially brings about a state in Y' for every $Y' \supseteq Y$.

The semantics of formulas and games is then defined by simultaneously extending V and N to non-atomic cases:

$$\begin{aligned} V_\bot &= \emptyset & N_{\alpha;\beta}(X) &= N_\alpha(N_\beta(X))\\ V_{\neg\varphi} &= \overline{V_\varphi} & N_{\alpha^{\mathsf{d}}}(X) &= \overline{N_\alpha(\overline{X})}\\ V_{\varphi\vee\psi} &= V_\varphi \cup V_\psi & N_{\alpha\cup\beta}(X) &= N_\alpha(X) \cup N_\beta(X)\\ V_{\langle\gamma\rangle\varphi} &= N_\gamma(V_\varphi) & N_{\varphi?}(X) &= V_\varphi \cap X\\ & & N_{\alpha^*}(X) &= \bigcap\{Y \subseteq S \mid X \cup N_\alpha(Y) \subseteq Y\}\end{aligned}$$

By induction, all N_α can be shown to be monotonic, and hence the operation $f_{\alpha,X}(Y) = X \cup N_\alpha(Y)$ will be monotonic as well. Thus, by the Knaster-Tarski theorem, $N_{\alpha^*}(X)$ is the least fixpoint of $f_{\alpha,X}(Y)$:

$$N_{\alpha^*}(X) = \mu Y.f_{\alpha,X}(Y) = \mu Y.X \cup N_\alpha(Y)$$

i.e. $f_{\alpha,X}(N_{\alpha^*}(X)) = N_{\alpha^*}(X)$ and for every $Z \subseteq S$, if $f_{\alpha,X}(Z) = Z$ then $N_{\alpha^*}(X) \subseteq Z$.

Finally, we say that φ is true in $\mathcal{I} = (S, \{N_g | g \in \Gamma_0\}, \{V_p | p \in \Phi_0\})$ at $s \in S$ (notation: $\mathcal{I}, s \models \varphi$) iff $s \in V_\varphi$. Some more standard terminology: φ is valid in $\mathcal{I}$ (denoted as $\mathcal{I} \models \varphi$) iff $V_\varphi = S$, and φ is valid (denoted as $\models \varphi$) iff it is valid in all models. φ and ψ are equivalent iff $\models \varphi \leftrightarrow \psi$. Lastly, ψ is a (global) consequence of φ (denoted as $\varphi \models \psi$) iff for all models $\mathcal{I}$, if $\mathcal{I} \models \varphi$ then $\mathcal{I} \models \psi$.

3 Bisimulation for Game Models

Bisimulation provides an answer to the question when two models or processes should be considered the same. Different criteria may come to mind depending on what aspects of the models one is interested in. If only interested in observable properties of processes, one may choose for finite-trace equivalence, but if interested in mathematical structure, one may choose isomorphism. These equivalence notions (see e.g. [7] for an overview) partition the class of models into equivalence classes, and one may order equivalence notions according to how fine-grained the induced partition is. While finite-trace equivalence is often considered as too coarse and isomorphism as too fine, bisimulation is situated between these two extremes.

As it stands, bisimulation cannot be applied to the game models of *GL* since these models are not processes. As will be discussed in the next section, the following definition generalizes the standard notion of bisimulation to the more general models used for *GL*. In a different context, this modification of bisimulation has been proposed to deal with concurrency in [4].

Definition 1 (Bisimulation). *Let $\mathcal{I} = (S, \{N_g | g \in \Gamma_0\}, \{V_p | p \in \Phi_0\})$ and $\mathcal{I}' = (S', \{N'_g | g \in \Gamma_0\}, \{V'_p | p \in \Phi_0\})$ be two models. Then $\sim \subseteq S \times S'$ is a bisimulation between $\mathcal{I}$ and $\mathcal{I}'$ iff for any $s \sim s'$ we have*

1. *For all $p \in \Phi_0$: $s \in V_p$ iff $s' \in V'_p$*
2. *For all $g \in \Pi_0$: If $s \in N_g(X)$ then $\exists X' \subseteq S'$ such that $s' \in N'_g(X')$ and $\forall x' \in X' \, \exists x \in X : x \sim x'$.*
3. *For all $g \in \Pi_0$: If $s' \in N'_g(X')$ then $\exists X \subseteq S$ such that $s \in N_g(X)$ and $\forall x \in X \, \exists x' \in X' : x \sim x'$.*

Two states $s \in S$ and $s' \in S'$ are bisimilar iff there is a bisimulation $\sim$ such that $s \sim s'$. If we want to make the underlying models explicit, we will write $(\mathcal{I}, s) \sim (\mathcal{I}', s')$.

The notions of invariance and safety generalize the bisimulation clauses from atomic to general formulas and games.

Definition 2 (*GL*-Invariance & Safety). *A GL-formula φ is invariant for bisimulation if for all models $\mathcal{I}$ and $\mathcal{I}'$, $(\mathcal{I}, s) \sim (\mathcal{I}', s')$ implies $\mathcal{I}, s \models \varphi \Leftrightarrow \mathcal{I}', s' \models \varphi$. A GL-game γ is safe for bisimulation if for all models $\mathcal{I}$ and $\mathcal{I}'$, $(\mathcal{I}, s) \sim (\mathcal{I}', s')$ implies (1) if $s \in N_\gamma(X)$ then $\exists X' \subseteq S'$ such that $s' \in N'_\gamma(X')$*

and $\forall x' \in X' \exists x \in X : x \sim x'$*, and (2) if* $s' \in N'_\gamma(X')$ *then* $\exists X \subseteq S$ *such that* $s \in N_\gamma(X)$ *and* $\forall x \in X \exists x' \in X' : x \sim x'$*.*

As an equivalence notion for game models, bisimulation requires that if Angel can guarantee φ in game g in one model, he must be able to guarantee something at least as strong in the other model. If this were not the case, the two models could be distinguished by playing g, since Angel can achieve more in one model than in the other. The following result shows that *GL* is sound for bisimulation equivalence, i.e. not too expressive: Bisimilar states cannot be distinguished by formulas of the language (invariance), and the game constructions provided do not produce games which can distinguish bisimilar states either (safety).

Proposition 1. *All GL-formulas are invariant for bisimulation, and all GL-games are safe for bisimulation.*

Proof. We prove invariance and safety by simultaneous induction on Φ and Γ. By definition, atomic games (formulas) are safe (invariant) for bisimulation.Consider two models $\mathcal{I} = (S, \{N_g | g \in \Gamma_0\}, \{V_p | p \in \Phi_0\})$ and $\mathcal{I}' = (S', \{N'_g | g \in \Gamma_0\}, \{V'_p | p \in \Phi_0\})$. For non-atomic formulas, the boolean cases are immediate and we shall only show one direction of invariance for $\langle\gamma\rangle\varphi$. If $\mathcal{I}, s \models \langle\gamma\rangle\varphi$, $s \in N_\gamma(V_\varphi)$ and so (by safety induction hypothesis for γ) there is some X' such that $s' \in N'_\gamma(X')$ and for all $x' \in X'$ there is some $x \in V_\varphi$ such that $x \sim x'$. By invariance induction hypothesis for φ, this means that $X' \subseteq V'_\varphi$, and so by monotonicity, $s' \in N'_\gamma(V'_\varphi)$, which establishes that $\mathcal{I}', s' \models \langle\gamma\rangle\varphi$.

As for proving that the game constructions of *GL* are safe for bisimulation, consider first the case of test $\varphi?$: If $s \in N_{\varphi?}(X) = V_\varphi \cap X$, let $X' := \{x' | \exists x \in X : x \sim x'\}$, where $\sim$ denotes the bisimulation as usual. Then $s' \in N'_{\varphi?}(X')$ by induction hypothesis (1.) for φ, and for all $x' \in X'$ there is some $x \in X$ such that $x \sim x'$, simply by definition of X'.

For union, if $s \in N_{\alpha\cup\beta}(X)$ we can assume w.l.o.g. that $s \in N_\alpha(X)$ and apply the induction hypothesis, i.e. for some X', we have $s' \in N'_\alpha(X')$ and hence also $s' \in N'_{\alpha\cup\beta}(X')$.

For composition, suppose that $s \in N_\alpha(N_\beta(X))$. Using the induction hypothesis for α, there is some Y' such that $s' \in N'_\alpha(Y')$ and for all $y' \in Y'$ there is a $u \in N_\beta(X)$ such that $u \sim y'$. Now let $X' := \{x' | \exists x \in X : x \sim x'\}$. We must show that $s' \in N'_\alpha(N'_\beta(X'))$. For this, it suffices by monotonicity to show that $Y' \subseteq N'_\beta(X')$. So suppose that $y' \in Y'$, i.e. for some $u \in N_\beta(X)$ we have $u \sim y'$. Using the induction hypothesis for β, there is some V' such that $y' \in N'_\beta(V')$ and for all $v' \in V'$ there is some $x \in X$ such that $x \sim v'$. Hence $V' \subseteq X'$ and so by monotonicity, $y' \in N'_\beta(X')$

Dual: Suppose $s \in N_{\alpha^d}(X)$, i.e. $s \notin N_\alpha(\overline{X})$. Again, let $X' := \{x' | \exists x \in X : x \sim x'\}$. It is sufficient to show that $s' \notin N'_\alpha(\overline{X'})$. Suppose by reductio the contrary. Then there is some Z with $s \in N_\alpha(Z)$ and for all $z \in Z$ there is some $x' \notin X'$ such that $z \sim x'$. From this it follows that $Z \subseteq \overline{X}$, so by monotonicity $s \in N_\alpha(\overline{X})$, a contradiction.

Iteration: Let $X' := \{x' | \exists x \in X : x \sim x'\}$ and $Z := \{z | \forall z' : z \sim z' \Rightarrow z' \in N'_{\alpha^*}(X')\}$. It is sufficient to show that $N_{\alpha^*}(X) \subseteq Z$, and given the definition of

$N_{\alpha^*}(X)$ as a least fixpoint, it suffices to show that $X \cup N_\alpha(Z) \subseteq Z$. Supposing that $x \in X$ and for some x' we have $x \sim x'$, we have $x' \in X' \subseteq N'_{\alpha^*}(X')$. On the other hand, suppose that $x \in N_\alpha(Z)$ and $x \sim x'$. Then by induction hypothesis, there is some Z' such that $x' \in N'_\alpha(Z')$ and for all $z' \in Z'$ there is some $z \in Z$ such that $z \sim z'$. But then $Z' \subseteq N'_{\alpha^*}(X')$, and so by monotonicity $x' \in N'_\alpha(N'_{\alpha^*}(X')) \subseteq N'_{\alpha^*}(X')$ which completes the proof. □

4 Games on Kripke Models I: μ-Calculus

In the remaining part of this paper, we shall look at a special class of game models, namely Kripke models. A Kripke model $\mathcal{I} = (S, \{R_g | g \in \Gamma_0\}, \{V_p | p \in \Phi_0\})$ differs from a game model in providing an accessibility relation $R_g \subseteq S \times S$ for every atomic game $g \in \Gamma_0$. In Kripke models, atomic games are particularly simple since they are 1-player games. For each atomic game, all choices within that game are made by Angel so that Angel has complete freedom in determining which terminal position will be reached. Thus, sR_gt will hold if when playing game g at state s, t is a possible final state. To obtain the corresponding game model from a Kripke model, let $N_g(X) = \{s \in S | \exists t \in X : sR_gt\}$. Under this correspondence, one can easily verify that for Kripke models, definition 1 indeed reduces to the following standard notion of bisimulation:

Definition 3 (Bisimulation for Kripke models). *Let $\mathcal{I} = (S, \{R_g | g \in \Gamma_0\}, \{V_p | p \in \Phi_0\})$ and $\mathcal{I}' = (S', \{R'_g | g \in \Gamma_0\}, \{V'_p | p \in \Phi_0\})$ be two Kripke models. Then $\sim \subseteq S \times S'$ is a bisimulation between $\mathcal{I}$ and $\mathcal{I}'$ iff for any $s \sim s'$ we have*

1. *For all $p \in \Phi_0$: $s \in V_p$ iff $s' \in V'_p$*
2. *For all $g \in \Gamma_0$: If sR_gt, then there is a $t' \in S'$ such that $s'R'_gt'$ and $t \sim t'$.*
3. *For all $g \in \Gamma_0$: If $s'R'_gt'$, then there is a $t \in S$ such that sR_gt and $t \sim t'$.*

Two well-known languages for describing Kripke models are *PDL* and the modal μ-calculus. The language of *PDL* differs from the language of *GL* only in not having the dual-operator available. Since this operator was responsible for introducing interaction between the players, all games which can be constructed within *PDL* will be 1-player games, i.e. nondeterministic programs.

The μ-calculus introduces fixpoint operators into the modal language, yielding a logic which is strictly more expressive than *PDL* (see [15]). Besides propositional constants Φ_0, the language contains propositional variables $X, Y, \ldots \in Var$ and the set of formulas is defined inductively as

$$\varphi := \bot \mid p \mid X \mid \neg\varphi \mid \varphi \vee \varphi \mid \langle\gamma_0\rangle\varphi \mid \mu X.\varphi$$

where $p \in \Phi_0$, $\gamma_0 \in \Gamma_0$, $X \in Var$ and in $\mu X.\varphi$, X occurs *strictly positively* in φ, i.e. every free occurrence of X in φ occurs under an even number of negations. Note that in contrast to *GL*, modalities are always atomic in the μ-calculus.

Formulas of the μ-calculus are interpreted over Kripke models as before (using the corresponding game model), but a variable assignment $v : Var \to \mathcal{P}(S)$ is needed to interpret variables. The semantics of the fixpoint formula is given by

$$V^v_{\mu X.\varphi} = \bigcap \{T \subseteq S | V^{v[X:=T]}_\varphi \subseteq T\}$$

where $V_\varphi^{v[X:=T]}$ differs from V_φ^v in assigning T to variable X. Since φ was assumed to be strictly positive in X, monotonicity is guaranteed and $\mu X.\varphi$ denotes the least fixpoint of the operation associated with $\varphi(X)$.

Inspecting the semantics of *GL*, one can easily translate *GL*-formulas into equivalent μ-calculus formulas, demonstrating that *GL* is a variable-free fragment of the μ-calculus. While a characterization of the precise expressiveness of this fragment is still lacking, some preliminary observations can be made: *GL* is strictly more expressive than *PDL*, since *GL* can express the existence of an infinite a-path by the formula

$$\mu X.[a]X \;=\; \langle (a^d)^* \rangle \bot$$

which cannot be expressed in *PDL* (see [15]). More complex properties such as "on some path p occurs infinitely often" ($EF^\infty p$ in CTL^* notation) can also be expressed (we assume that $\Gamma_0 = \{a\}$):

$$\nu X.\mu Y.\langle a \rangle((p \wedge X) \vee Y) = [((a^*; a; p?)^d)^*]\top$$

where $\nu X.\varphi$ abbreviates $\neg\mu X.\neg\varphi(\neg X)$ and yields the greatest fixpoint of $\varphi(X)$. More generally, if we let $g_0 = a$ and $g_{n+1} = (g_n^d)^*$, the μ-calculus translation of $\langle g_n \rangle \bot$ will be a formula of alternation depth n, so that *GL* formulas cover all levels of the alternation hierarchy as defined in [10].

5 Games on Kripke Models II: First-Order Logic

It is well-known that modal logic and *PDL* without iteration can be translated into *FOL*. In spite of the second-order appearance of Game Logic, a translation into *FOL* is possible here as well: The signature contains a unary relation symbol V_p for every propositional letter $p \in \Phi_0$, and a binary relation symbol R_g for every atomic game $g \in \Gamma_0$. Furthermore, we allow for second-order variables $\mathsf{X}, \mathsf{Y}, \ldots$ as well. Thus, the unary relation symbols now comprise constants as well as variables. As will become clear later, we will not quantify over these variables but only use them as a matter of convenience to serve as place-holders for substitution; hence, we can still consider the language to be first-order. We define the translation function $^\circ$ which maps a *GL*-formula φ to a *FOL*-formula with one free variable x, and an iteration-free *GL*-game γ to a *FOL*-formula with two free variables x and Y.

$$\begin{aligned}
p^\circ &= \mathsf{V}_p\mathsf{x} \quad \text{for } p \in \Phi_0 & g^\circ &= \exists \mathsf{z}(\mathsf{x}\mathsf{R}_g\mathsf{z} \wedge \mathsf{Y}\mathsf{z}) \quad \text{for } g \in \Gamma_0 \\
(\neg\varphi)^\circ &= \neg\varphi^\circ & (\varphi?)^\circ &= \varphi^\circ \wedge \mathsf{Y}\mathsf{x} \\
(\varphi \vee \psi)^\circ &= \varphi^\circ \vee \psi^\circ & (\alpha \cup \beta)^\circ &= \alpha^\circ \vee \beta^\circ \\
(\langle\gamma\rangle\varphi)^\circ &= \gamma^\circ[\mathsf{Y} := \varphi^\circ] & (\alpha; \beta)^\circ &= \alpha^\circ[\mathsf{Y} := \beta^\circ] \\
& & (\alpha^\mathsf{d})^\circ &= \neg\alpha^\circ[\mathsf{Y} := \neg\mathsf{Y}\mathsf{x}]
\end{aligned}$$

In this definition, substitution for second-order variables is used as follows: Given two *FOL*-formulas δ and ξ where ξ contains exactly one free first-order variable, say x, $\delta[\mathsf{Y} := \xi]$ denotes the result of replacing every occurrence $\mathsf{Y}t$ in

δ by $\xi[\mathsf{x} := t]$. As an example, $\exists \mathsf{z}(\mathsf{x}\mathsf{R}_g\mathsf{z} \wedge \mathsf{Y}\mathsf{z})[\mathsf{Y} := \neg\mathsf{Y}\mathsf{x}]$ yields $\exists \mathsf{z}(\mathsf{x}\mathsf{R}_g\mathsf{z} \wedge \neg\mathsf{Y}\mathsf{z})$. Some more remarks on notation: $\varphi(x_1, \ldots, x_n)$ refers to a formula φ whose free variables (first- and second-order) are among $x_1, \ldots, x_n$. When a formula has been introduced in this way, $\varphi(t_1, \ldots, t_n)$ denotes $\varphi[x_1 := t_1, \ldots, x_n := t_n]$, i.e. the simultaneous substitution of t_i for x_i in φ.

Regarding the semantics, we can interpret a Kripke model $\mathcal{I} = (S, \{R_g | g \in \Gamma_0\}, \{V_p | p \in \Phi_0\})$ as a first-order model in the obvious way, taking R_g as the interpretation of R_g, and interpreting V_p as V_p. For a unary predicate symbol V_p and $X \subseteq S$, let $\mathcal{I}_{p:=X}$ be the model which is the same as $\mathcal{I}$ except that $V_p = X$. Given a model $\mathcal{I}$, states $s_1, \ldots, s_m \in S$, sets of states $S_1, \ldots, S_n \subseteq S$ and a *FOL*-formula $\varphi(\mathsf{x}_1, \ldots, \mathsf{x}_m, \mathsf{X}_1, \ldots, \mathsf{X}_n)$, we write $\mathcal{I} \models \varphi[s_1, \ldots, s_m, S_1, \ldots, S_n]$ to denote that φ is true in $\mathcal{I}$ according to the standard *FOL* semantics when x_i is assigned the value s_i and X_i the value S_i.

The following result states the semantic correctness of the translation function.

Lemma 1. *For all GL-formulas φ, games γ and Kripke models $\mathcal{I} = (S, \{R_g | g \in \Gamma_0\}, \{V_p | p \in \Phi_0\})$: $\mathcal{I}, s \models \varphi$ iff $\mathcal{I} \models \varphi^\circ[s]$ and $s \in N_\gamma(X)$ iff $\mathcal{I} \models \gamma^\circ[s, X]$.*

As with the safety result for program constructions, the safety result for game constructions makes use of the characterization of the modal fragment of *FOL* as its bisimulation-invariant fragment. The definition of invariance and safety (definition 2) which was phrased for *GL* has its natural first-order analogue:

Definition 4 (*FOL*-Invariance & Safety). *A FOL-formula $\varphi(\mathsf{x})$ is invariant for bisimulation if for all models $\mathcal{I}$ and $\mathcal{I}'$, $(\mathcal{I}, s) \sim (\mathcal{I}', s')$ implies that $\mathcal{I} \models \varphi[s]$ iff $\mathcal{I}' \models \varphi[s']$. A first-order formula $\varphi(\mathsf{x}, \mathsf{Y})$ is safe for bisimulation if for all models $\mathcal{I}$ and $\mathcal{I}'$, $(\mathcal{I}, s) \sim (\mathcal{I}', s')$ implies (1) if $\mathcal{I} \models \varphi[s, T]$ then there is some T' such that $\mathcal{I}' \models \varphi[s', T']$ and for all $t' \in T'$ there is some $t \in T$ such that $t \sim t'$, and (2) if $\mathcal{I}' \models \varphi[s', T']$ then there is some T such that $\mathcal{I} \models \varphi[s, T]$ and for all $t \in T$ there is some $t' \in T'$ such that $t \sim t'$.*

By a *modal formula* we mean a *GL*-formula which only contains atomic games (i.e. also no tests). The classic result from [5] can now be stated as follows:

Lemma 2. *A FOL-formula $\varphi(\mathsf{x})$ is invariant for bisimulation iff it is equivalent to the translation of a modal formula.*

For the rest of this section, we will assume that games are iteration-free. Call a *FOL*-formula $\varphi(\mathsf{x}, \mathsf{Y})$ *monotonic* iff for all Kripke models $\mathcal{I}$ and states s, $\mathcal{I} \models \varphi[s, X]$ implies $\mathcal{I} \models \varphi[s, X']$ for every $X \subseteq X'$. Similarly, call a modal formula φ *monotonic in p* iff for all Kripke models $\mathcal{I}$ and states s, $\mathcal{I}_{p:=X}, s \models \varphi$ implies $\mathcal{I}_{p:=X'}, s \models \varphi$ for every $X \subseteq X'$. Lastly, let $Pos(\varphi)$ ($Neg(\varphi)$) be the set of atomic propositions which occur positively (negatively) in φ, i.e. under an even (odd) number of negations. Thus, formula φ is strictly positive (negative) in p iff $p \notin Neg(\varphi)$ ($p \notin Pos(\varphi)$).

The final lemma needed relates the syntactic notion of positivity to the semantic notion of monotonicity. It makes use of the Lyndon interpolation theorem for modal logic (see e.g. [17]) and the global deduction theorem (taken from [11]).

Lemma 3 (Lyndon Interpolation Theorem). *If $\models \alpha \to \beta$ for modal formulas α, β, then there exists a modal formula γ such that (1) $\models \alpha \to \gamma$, (2) $\models \gamma \to \beta$, (3) $Pos(\gamma) \subseteq Pos(\alpha) \cap Pos(\beta)$, and (4) $Neg(\gamma) \subseteq Neg(\alpha) \cap Neg(\beta)$.*

Lemma 4 (Global Deduction Theorem). *For modal formulas δ and γ, $\delta \models \gamma$ iff there is some $n > 0$ such that $\models (\Box^1\delta \wedge \ldots \wedge \Box^n\delta) \to \gamma$, where each $\Box^i$ represents a possibly empty sequence of universal modalities labeled by (possibly different) atomic games.*

Lemma 5. *A modal formula φ is monotonic in p iff it is equivalent to a modal formula strictly positive in p.*

Proof. One can easily check by induction that strictly positive modal formulas are monotonic, so we shall only prove the other direction. If $\varphi(p)$ is monotonic in p, then taking a proposition letter q not occurring in φ, we have $p \to q \models \varphi(p) \to \varphi(q)$ (recall that semantic consequence was defined globally). By lemma 4, we know that

$$(\Box^1(p \to q) \wedge \ldots \wedge \Box^n(p \to q)) \to (\varphi(p) \to \varphi(q))$$

is valid, and as a consequence,

$$\varphi(p) \to ((\Box^1(p \to q) \wedge \ldots \wedge \Box^n(p \to q)) \to \varphi(q))$$

is also valid. By lemma 3, this implies that

$$\varphi(p) \to \gamma \quad \text{and} \quad \gamma \to ((\Box^1(p \to q) \wedge \ldots \wedge \Box^n(p \to q)) \to \varphi(q))$$

are valid, for some modal formula γ which does not contain q and which is strictly positive in p. The second conjunct implies that $\gamma \to \varphi(p)$ is valid: For suppose $\mathcal{I}, s \models \gamma$ and $X = \{t | \mathcal{I}, t \models p\}$. Then since γ does not contain q, $\mathcal{I}_{q:=X}, s \models \gamma$. From this it follows that $\mathcal{I}_{q:=X}, s \models \varphi(q)$ and hence $\mathcal{I}, s \models \varphi(p)$. Thus, φ is equivalent to γ, a modal formula strictly positive in p. □

The main lemma we need for our safety result relates monotonic modal formulas to *GL*-formulas of a special kind.

Lemma 6. *Every modal formula φ which is monotonic in p is equivalent to a GL-formula $\langle\gamma\rangle p$, where γ is a game which does not contain p.*

Proof. We prove by induction that every modal formula φ which is strictly positive (negative) in p is equivalent to a *GL*-formula $\langle\gamma\rangle p$ ($\neg\langle\gamma\rangle p$), where γ does not contain p. Then the result follows by lemma 5. The following table provides the equivalent *GL*-formulas for every modal formula φ depending on whether φ is strictly positive or strictly negative in p.

modal formula	str. pos/neg	*GL*-formula	ind. hyp.
p	*pos*	$\langle \top ? \rangle p$	–
$q \neq p$	*pos*	$\langle q?; \bot ?^{\mathsf{d}} \rangle p$	–
$q \neq p$	*neg*	$\neg \langle q?^{\mathsf{d}}; \bot ? \rangle p$	–
$\neg \varphi$	*pos*	$\langle \gamma \rangle p$	$\models \varphi \leftrightarrow \neg \langle \gamma \rangle p$
$\neg \varphi$	*neg*	$\neg \langle \gamma \rangle p$	$\models \varphi \leftrightarrow \langle \gamma \rangle p$
$\varphi_1 \vee \varphi_2$	*pos*	$\langle \gamma_1 \cup \gamma_2 \rangle p$	$\models \varphi_i \leftrightarrow \langle \gamma_i \rangle p$
$\varphi_1 \vee \varphi_2$	*neg*	$\neg \langle (\gamma_1{}^{\mathsf{d}} \cup \gamma_2{}^{\mathsf{d}})^{\mathsf{d}} \rangle p$	$\models \varphi_i \leftrightarrow \neg \langle \gamma_i \rangle p$
$\langle g \rangle \varphi$	*pos*	$\langle g; \gamma \rangle p$	$\models \varphi \leftrightarrow \langle \gamma \rangle p$
$\langle g \rangle \varphi$	*neg*	$\neg \langle g^{\mathsf{d}}; \gamma \rangle p$	$\models \varphi \leftrightarrow \neg \langle \gamma \rangle p$

□

Proposition 2. *A FOL-formula $\varphi(\mathsf{x}, \mathsf{Y})$ is equivalent to the translation of a GL-game iff it is safe for bisimulation and monotonic in* Y.

Proof. If $\varphi(\mathsf{x}, \mathsf{Y})$ is equivalent to the translation of a *GL*-game γ, then using lemma 1, φ will be monotonic in Y (because N_γ is monotonic) and safe for bisimulation (by proposition 1).

For the converse, assume that $\varphi(\mathsf{x}, \mathsf{Y})$ is monotonic and safe for bisimulation. Taking a new predicate symbol V_p which does not occur in φ, $\varphi(\mathsf{x}, \mathsf{V}_p)$ will be invariant for bisimulation. By lemma 2, $\varphi(\mathsf{x}, \mathsf{V}_p)$ is equivalent to the translation of a modal formula δ, i.e. $\models \varphi(\mathsf{x}, \mathsf{V}_p) \leftrightarrow \delta^\circ$. Since $\varphi(\mathsf{x}, \mathsf{Y})$ was monotonic, δ will be monotonic in p and by lemma 6, $\models \delta \leftrightarrow \langle \gamma \rangle p$ where γ is a *GL*-game which does not contain p, and so $\models \varphi(\mathsf{x}, \mathsf{V}_p) \leftrightarrow (\langle \gamma \rangle p)^\circ$. It can now be checked that $\models \varphi(\mathsf{x}, \mathsf{Y}) \leftrightarrow \gamma^\circ$: If $\mathcal{I} \models \varphi[s, X]$ then given that V_p does not occur in φ, $\mathcal{I}_{p:=X} \models \varphi(\mathsf{x}, \mathsf{V}_p)[s]$ and so $\mathcal{I}_{p:=X} \models (\langle \gamma \rangle p)^\circ[s]$. Since p does not occur in γ, this implies that $\mathcal{I} \models \gamma^\circ[s, X]$. The converse is proved along the same lines. □

On the one hand, proposition 2 provides a characterization result for the iteration-free games which can be constructed in Game Logic: *GL*-games are the monotonic bisimulation-safe formulas $\varphi(\mathsf{x}, \mathsf{V}_p)$ of first-order logic (we can simply replace the variable Y by a designated unary predicate constant V_p). In other words, the game-fragment of *FOL* is precisely the monotonic bisimulation-safe fragment. On the other hand, looking at the set of operations on games which *GL* provides, one may ask whether one could not add other natural operations to create new games (e.g. playing games in parallel), thus increasing the expressive power of the language. Proposition 2 demonstrates that if the new game operation is (1) first-order definable, (2) monotonic and (3) safe for bisimulation, then it is expressible in *GL* already. As argued before, requirements (2) and (3) are natural desiderata for games, i.e. they are minimal requirements for any alleged game operation, and so the operations of test, union, composition and dual are sufficient to construct all first-order definable games.

The result concerning bisimulation-safe programs from [6] can be reformulated to fit the present framework. Semantically, the difference between games

and programs lies in the difference between monotonicity and continuity: Call a *FOL*-formula $\varphi(\mathsf{x}, \mathsf{Y})$ *continuous* iff for all Kripke models $\mathcal{I}$ and states s, $\mathcal{I} \models \varphi[s, \bigcup_{X \in V} X]$ iff there is some $X \in V$ for which $\mathcal{I} \models \varphi[s, X]$ holds. Then the program-analogue of proposition 2 states that a *FOL*-formula $\varphi(\mathsf{x}, \mathsf{Y})$ is equivalent to the translation of a *GL*-program iff it is safe for bisimulation and continuous in Y, where *GL*-programs are dual-free *GL*-games. Thus, the dual operator makes all the difference between programs and games; without dual, we obtain all first-order definable programs, with dual, all first-order definable games.

6 Beyond First-Order Logic

The last two sections were concerned with Kripke models rather than game models in general. The reason for this restriction is that game models are rather unorthodox structures. We do not know of any logical languages besides non-normal modal logics and Game Logic which have been proposed for these structures. Consequently, this prevents an easy extension of the definability result of proposition 2 to *GL* over general game models.

Even for Kripke models, the translation into *FOL* carried out in the previous section relied on the restriction to iteration-free games. For programs, a stronger definability result covering iteration has been obtained in [13] which characterizes the class of monadic-second-order definable programs which are safe for bisimulation. The proof makes use of the fact that the bisimulation-invariant fragment of monadic second-order logic is the μ-calculus [14]. An extension of proposition 2 along these lines however would require a better understanding of how exactly *GL* relates to the μ-calculus. As for the μ-calculus itself, many fundamental properties were established only recently, such as completeness [19], the non-collapse of the alternation-hierarchy [8] and uniform interpolation [1], and others such as Lyndon interpolation are still open.

To summarize, the restriction of the scope of proposition 2 to *FOL* is due to the fact that *FOL* is one of the logics we know most about and is able to express the most fundamental game-operations. When moving to stronger languages one has different options available, always depending on the game constructions one is interested in. For besides playing a game iteratively, playing two games in parallel or interleaved might present another attractive game construction worth investigating in relation to bisimulation.

References

1. Giovanna D'Agostino. *Modal Logic and non-well-founded Set Theory: translation, bisimulation, interpolation.* PhD thesis, University of Amsterdam, 1998.
2. Ralph-Johan Back and Joakim von Wright. *Refinement Calculus - A systematic introduction.* Springer, 1998.
3. R.J.R. Back and J. von Wright. Games and winning strategies. *Information Processing Letters*, 53:165–172, 1995.
4. J. van Benthem, J. van Eijck, and V. Stebletsova. Modal logic, transition systems and processes. Computer Science Report CS-R9321, CWI, 1993.

5. Johan van Benthem. *Modal Correspondence Theory*. PhD thesis, University of Amsterdam, 1976.
6. Johan van Benthem. Program constructions that are safe for bisimulation. *Studia Logica*, 60(2):311–330, 1998.
7. Johan van Benthem and Jan Bergstra. Logic of transition systems. ILLC Prepublication Series CT-93-03, University of Amsterdam, 1993.
8. J. C. Bradfield. The modal mu-calculus alternation hierarchy is strict. In U. Montanari and V. Sassone, editors, *Proceedings of CONCUR '96*, volume 1119 of *LNCS*, pages 233–246. Springer, 1996.
9. Brian Chellas. *Modal Logic - An Introduction*. Cambridge University Press, 1980.
10. E. Allen Emerson and Chin-Laung Lei. Efficient model checking in fragments of the propositional mu-calculus. In *Proceedings of the 1st IEEE Symposium on Logic in Computer Science*, pages 267–278, 1986.
11. Melvin Fitting. Basic modal logic. In D. Gabbay, C. Hogger, and J. Robinson, editors, *Handbook of logic in artificial intelligence and logic programming*, volume 1, pages 365–448. Oxford University Press, 1993.
12. David Harel. Dynamic logic. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic*, volume II. D. Reidel Publishing Company, 1984.
13. Marco Hollenberg. *Logic and Bisimulation*. PhD thesis, University of Utrecht, 1998.
14. David Janin and Igor Walukiewicz. On the expressive completeness of the propositional mu-calculus with respect to monadic second order logic. In *Proceedings of CONCUR '96*, volume 1119 of *LNCS*, pages 263–277. Springer, 1996.
15. Dexter Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.
16. Dexter Kozen and Jerzy Tiuryn. Logics of programs. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B. MIT Press, 1990.
17. Larisa Maksimova. Amalgamation and interpolation in normal modal logics. *Studia Logica*, 50(3-4):457–471, 1991.
18. Rohit Parikh. The logic of games and its applications. In Marek Karpinski and Jan van Leeuwen, editors, *Topics in the Theory of Computation*, volume 24 of *Annals of Discrete Mathematics*. Elsevier, 1985.
19. Igor Walukiewicz. A note on the completeness of Kozen's axiomatisation of the propositional μ-calculus. *Bulletin of Symbolic Logic*, 2(3):349–366, 1996.

Logical Relations and Data Abstraction

John Power*,[1] and Edmund Robinson**,[2]

[1] Laboratory for the Foundations of Computer Science, University of Edinburgh
ajp@dcs.ed.ac.uk
[2] Department of Computer Science, Queen Mary and Westfield College,
edmundr@dcs.qmw.ac.uk

Abstract. We prove, in the context of simple type theory, that logical relations are sound and complete for data abstraction as given by equational specifications. Specifically, we show that two implementations of an equationally specified abstract type are equivalent if and only if they are linked by a suitable logical relation. This allows us to introduce new types and operations of any order on those types, and to impose equations between terms of any order. Implementations are required to respect these equations up to a general form of contextual equivalence, and two implementations are equivalent if they produce the same contextual equivalence on terms of the enlarged language. Logical relations are introduced abstractly, soundness is almost automatic, but completeness is more difficult, achieved using a variant of Jung and Tiuryn's logical relations of varying arity. The results are expressed and proved categorically.

Keywords: logical relations, cartesian closed fibrations, interpretations, lambda calculus

1 Introduction

Logical relations are a standard tool for establishing the equivalence of data representations. If one can find a family of relations on corresponding types which is a congruence, i.e., are preserved by operations of the theory, and reduces to equality on observable types, then clearly observable operations will be equal in the two representations. Such a family need not form a logical relation (it may form a lax or pre-logical relation [4,12]), but logical relations do provide a way to construct such families by induction on type structure. So the establishment of a logical relation suffices to determine equivalence of representation at least for simple types. It has long been known that for algebraic theories with first order operations and equations, equivalent representations are indeed linked by a logical relation, see Mitchell [9,10]. It has also long been known that standard logical relations are not complete in this sense for higher-order theories.

* This work has been done with the support of EPSRC grant GR/M56333 and a British Council grant.

** This author would like to acknowledge the support of an EPSRC Advanced Fellowship, EPSRC grant GR/L54639, and ESPRIT Working Group 6811 Applied Semantics

P. Clote and H. Schwichtenberg (Eds.): CSL 2000, LNCS 1862, pp. 497–511, 2000.

The purpose of this paper is to show that by relaxing the definition of logical relation, we can extend Mitchell's completeness result to higher-order theories.

The approach we take here has two major sources. The first is the work of Achim Jung and Jerzy Tiuryn [5], who used a form of logical relations to characterise lambda definability in the simple type hierarchy. The second is the thesis of Claudio Hermida [3], which explores the connection between logical predicates and logical formalisms expressed as fibrations defined over the semantic category. Sadly there is still no really accessible reference for the material and ideas in this thesis.

From a practical angle, the notion of abstract type we use here is quite restricted as we only consider equationally specified types, albeit with equations between higher-order operators. This is also the level of generality of Alimohamed's work on definability [1]. However, it raises a natural question of whether the link with logic exploited in this paper could be further exploited to allow a less restrictive specification language.

Finally, if data abstraction is the study of equivalence of representation, then data refinement is the study of improvements. The two ideas are clearly linked, and the fundamental notion of this paper, that of being "linked by a logical relation", is closely related to the notion of L-relation in [6] (see also [14]). The work of that paper, which is also about data equivalence but from a purely categorical perspective, and the work of this paper, and those on lax and prelogical relations [4,12] clearly need to be combined at some point.

A natural question, however, is why we chose to base our work on cartesian closed categories rather than Henkin models.

Henkin models provide a natural extension of classical set-based first-order model theory to the higher-order setting of simply typed lambda calculi. Much of their appeal lies in the immediacy of this relation. Cartesian closed categories, on the other hand, arose out of the desire to express in purely category theoretic terms the characteristic properties of function spaces such as those found in Set. The formal link with typed lambda calculus, though not surprising in itself, came surprisingly late (cf. Lambek and Scott [7]), and prompted a slight redefinition of the notion to allow categories which had products but not arbitrary finite limits.

Cartesian closed categories and Henkin models are intimately linked. If one starts with a Henkin model of $\lambda\!\!\rightarrow$ one can easily extend it to model the type theory with products, $\lambda\!\times\!\!\rightarrow$ (using the fact that the type $(A_1 \times \ldots \times A_n) \rightarrow (B_1 \times \ldots \times B_m)$ is isomorphic to a product of pre-existing types $(A_1 \rightarrow (\ldots (A_n \rightarrow B_1) \ldots)) \times \ldots \times (A_1 \rightarrow (\ldots (A_n \rightarrow B_m) \ldots)))$. If one has a Henkin model of typed lambda calculus with products, then one can obtain a (concrete) cartesian closed category. The objects are types and the morphisms $A \longrightarrow B$ are the elements of the function type $[A \rightarrow B]$ in the Henkin model. Conversely if one has a cartesian closed category C, together with a product-preserving functor $U : C \longrightarrow \mathsf{Set}$, then one obtains an applicative structure with product types. The types are the objects of the category, and the type A is interpreted as the set UA. This automatically has representations of the combinators because of their presence

as morphisms in the cartesian closed category. However it is not necessarily a Henkin model because in general different elements of the function type may represent the same function. If, however, C is a concrete category and U its underlying set functor, this will not happen. In that case we do get a Henkin model.

So, from this perspective, cartesian closed categories generalise Henkin models. The question is whether that generality, or that difference in structure, is useful. We believe that it is.

One technical difference between the two approaches is apparent if one thinks of semantics taken in some kind of sets with structure (e.g. domains). It is natural to ask when this collection of sets forms a model of the simply typed lambda calculus. The category-theoretic answer is that one has to specify an appropriate notion of homomorphism. The resulting category is then either cartesian closed, in which case it supports a model structure, or it is not, when it doesn't. Moreover, if it is cartesian closed then the structure we use to model the lambda calculus is essentially unique. In contrast, it is much less clear what the constraints on a Henkin structure might be, or in what sense the structure would be unique if it existed. Essentially the development has to mimic the category-theoretic. First one defines an appropriate notion of homomorphism. Then one asks whether the set of homomorphisms between two objects can be made to carry the structure of the objects we are interested in. Or rather, since it almost always can, whether it can be made to do so in a way that makes application and the combinators homomorphisms. It is perhaps because of this apparently greater complexity that domain theorists and others talk about specific categories of domains being cartesian closed, and not about Henkin structures on classes of domains.

Moreover, Henkin models can only talk about models built on sets. This constraint is not always natural. For example it is natural to say that the canonical way of viewing a partially-ordered set as a category turns a Heyting (or Boolean) algebra into a cartesian closed category. It is not natural to say that they carry a Henkin model structure, since that requires presenting the elements of the partial order as themselves sets. A further example comes from the Curry-Howard correspondence. This can be conveniently and precisely expressed by saying that there is an isomorphism between a category whose objects are types and whose morphisms are $\beta\eta$-equivalence classes of λ-terms, and a category whose objects are propositions and whose morphisms are normalisation classes of proofs. The Henkin model account has to be forced to apply to this, most notably because in the event that A is not provable, then a proof of $A \Rightarrow B$ cannot adequately be described as a function which takes proofs of A to proofs of B.

We now turn to possible directions for our own future work. First, there are links between this work and the work of Honsell and Sannella on pre-logical relations. Unfortunately a preliminary account of these links had to be cut from this paper for reasons of space. We intend to return to this theme. An obvious direction in which to extend this work is to address models of call-by-value lambda-calculi such as the computational lambda-calculus [11].

Several people have been of considerable help in making comments or suggestions: they include Adam Eppendahl, Claudio Hermida, Peter O'Hearn, David Pym, Makoto Takeyama, and Bob Tennent. We would also like to thank the (many) anonymous referees of the original paper. We are sorry that we have not been able to make all the improvements they suggested.

2 Contextual Equivalence

In this section, we give a categorical account of a standard Morris-style contextual equivalence. As we are working in the context of simple type theory, and the theory may well be strongly normalising, we use a denotational rather than an operational approach, and observe values produced by computations rather than termination.

The basic structure is as follows. We assume a given cartesian closed category C. For instance, C may be the category Set of small sets, or, if we want to be sure to include fixed point operators to model recursion, we could consider C to be the category of ω-cpo's with least element, and with maps being functions that preserve the poset structure and sup's of ω-chains. Other alternatives for C include categories generated by axiomatic domain theory [2], or presheaf categories, or toposes. We simply require that C be cartesian closed. The category C will represent our base category in which we take our denotational semantics.

We further assume that we are given a set of observable pairs of types $Obs \subseteq ob(C) \times ob(C)$. The intuition behind this is that we are allowed to form judgements of equality

$$\boldsymbol{x} : A \vdash e = e' : B$$

where $(A, B) \epsilon Obs$. An alternative approach is to use a set of observable types, but we think this accords better with our open-term interpretation of the lambda calculus, and will allow a simpler extension to dependent type theory.

We use the structure of observables to generate a contextual equivalence $\approx$ on C, which allows us to compare $f \approx g$ when f and g are maps with common domain and codomain: type-theoretically, this amounts to demanding that contextually equivalent terms have the same type and are defined in the same context.

Example 1.
1. The underlying category is CPO, and $Obs = \{(1, 1_\perp)\}$. This gives one of the standard forms of Morris-style contextual equivalence, in which we are allowed to observe termination of closed terms.
2. The underlying category is Set, and $Obs = \{(1, Bool)\}$. Here, there is no non-termination, but contexts produce closed terms of type $Bool$. We can observe the boolean produced.
3. $Obs = ob(C) \times ob(C)$. In this case everything is observable. Not surprisingly $\approx$ is just equality.

Definition 1. *Given $f, g : X \longrightarrow Y$, we define $\sim$ by $f \sim g$ if for all $(A, B) \epsilon Obs$, and for all $\alpha : A \longrightarrow X$ and $\beta : Y \longrightarrow B$, we have*

$$A \xrightarrow{\alpha} X \xrightarrow{f} Y \xrightarrow{\beta} B$$

equals

$$A \xrightarrow{\alpha} X \xrightarrow{g} Y \xrightarrow{\beta} B$$

The category theoretic definition of a congruence is often used to mean an equivalence relation on each homset of a category C such that that equivalence relation is respected by composition on either side. Here, by a congruence, we mean that notion together with invariance under products and exponentials. That allows us to make the primary definition of this section as follows.

Definition 2. *We define $\approx$ to be the largest congruence contained in $\sim$.*

Mercifully, we can characterise $\approx$ more directly by means of the following proposition. We denote the exponential of X raised to the power U by $[U, X]$. We follow this convention for exponentials in a category, for exponentials of not only objects but maps, and for exponentials of categories, i.e., for functor categories, when we need them.

Proposition 1. *Given $f, g : X \longrightarrow Y$, the following are equivalent:*

1. *$f \approx g$*
2. *for all U, V in $ob(C)$, we have $[U, f] \times V \sim [U, g] \times V$*

To prove this proposition, apply the following lemma.

Lemma 1. *Let $\sim$ be an equivalence relation on parallel pairs of arrows of a category C. Then*

1. *composition preserves $\sim$, and hence $C/\sim$ is a category and $C \longrightarrow C/\sim$ is a functor if and only $f \sim g$ implies $\beta \cdot f \cdot \alpha \sim \beta \cdot g \cdot \alpha$.*
2. *if C has finite products, then $C/\sim$ has finite products, and passage to the quotient preserves them if and only if 1 holds and $f \sim g$ implies $f \times V \sim g \times V$ for all V in $ob(C)$.*
3. *if C is cartesian closed, then $C/\sim$ is a cartesian closed category and passage to the quotient is a cartesian closed functor if and only if 1 and 2 hold and $f \sim g$ implies $[U, f] \sim [U, g]$ for all U in $ob(C)$, or equivalently if and only if 1 holds and $f \sim g$ implies $[U, f] \times V \sim [U, g] \times V$ for all U,V in $ob(C)$.*

We can also make precise the relationship with Morris-style contextual equivalence. Any cartesian closed category can be regarded as a model of an applied lambda calculus in which we have base types corresponding to the objects of the category, and for each morphism $f : X \longrightarrow Y$, we are given a term constructor (function) $\hat{f}$, such that if e is a term of type X, then $f(e)$ is a term of type Y

(as in Lambek and Scott [7]). As usual, a context, $\mathcal{C}[-]$ is a term with a hole (written $[-]$), such that when we plug a term of suitable type into the hole, we get something that we can run and observe. Making this precise for typed open terms is always messy. The simplest solution is to consider terms in context. In that case, we demand

$$\frac{x : X \vdash e : Y}{a : A \vdash \mathcal{C}[e] : B}$$

should be an admissible rule for the type system (here, in order to make $\mathcal{C}[e]$ directly observable, $(A, B) \in Obs$). This can be achieved by ensuring that

$$\frac{x : X \vdash [-] : Y}{a : A \vdash \mathcal{C}[-] : B}$$

is a derived rule.

Definition 3. *Two terms e and e' (more accurately, two terms in the same context: $x : X \vdash e : Y$ and $x : X \vdash e' : Y$ ...) are contextually equivalent, $e \cong e'$ if and only if for all contexts $\mathcal{C}[-]$, $M\{\mathcal{C}[e]\} = M\{\mathcal{C}[e']\}$, where $M\{e\}$ is the semantics of e as taken in the category C.*

Proposition 2. *$e \cong e'$ if and only if $M\{e\} \approx M\{e'\}$.*

Proof. In the if direction, this is a simple structural induction on the context. In the other, it depends on the expressibility of substitution via abstraction, application, and beta reduction, and on the fact that all morphisms in the category are expressible as terms in the calculus.

3 Abstract Types

An equationally specified abstract type consists of a new type equipped with operations linking it to pre-existing types, and certain equations on terms built from those operations. The canonical example is stack. For our purposes, we may as well have a number of new types, and we interpret the operations as (open) terms in context

$$\boldsymbol{x} : \boldsymbol{X} \vdash f : Y$$

where the $\boldsymbol{X}$ and Y may be arbitrary type expressions built up from old and new types. This generates an extended lambda calculus in which the equations are interpreted as equality judgements

$$\boldsymbol{x} : \boldsymbol{X} \vdash e = e' : Y$$

Such a system freely generates a cartesian closed category E which comes with a cartesian closed functor $i : C \longrightarrow E$. This functor is injective on objects, but in general is neither full nor faithful as we can add extra operations between pre-existing types and impose new equations between pre-existing operations.

Conversely, given any injective on objects cartesian closed functor $i : C \longrightarrow E$, the category E is equivalent to a category constructed from C in this fashion, as we may generate it using all objects in $E \backslash i(C)$ and take all valid equations. So we can take $i : C \longrightarrow E$ as the extension of C by some abstract types.

We split this construction into two stages: first adjoin types and operations freely to give $i : C \longrightarrow D$, and then quotient by the equations to give $q : D \longrightarrow D/\approx_E$. In fact, we shall never use the free property of D and can take it to be an arbitrary cartesian closed category. The free property of such a D is fundamental to the work generalising logical relations for an account of data refinement in [6].

An implementation of the abstract type back in C is given by a cartesian closed functor $F : D \longrightarrow C$. We do not want to change the interpretation of the operations in C, so require $F \cdot i = id_C$. We also want the interpretation to validate the equations of the abstract type, at least up to a suitable notion of observational congruence. We illustrate the problems with the canonical simple (but first-order) example:

Example 2. Let C be Set and let D consist of Set together with the formal addition of a type stack together with its operations (empty, top, pop and push) and equations ($\mathsf{pop}(\mathsf{push}(n, s)) = s$, and $\mathsf{top}(\mathsf{push}(n, s)) = n$). An implementation amounts to a realisation of the type stack together with its operations and subject to its equations in Set in both cases. Consider an array implementation, in which stack is implemented by an (unbounded) array and a top pointer, and pop simply decrements the top pointer, leaving the array unchanged. In this implementation, the interpretation of $\mathsf{pop}(\mathsf{push}(n, s))$ is not equal to s. Moreover, in Set we can apply an operation which reads the stack one place above the top pointer, and hence observe the difference. Of course this violates the stack abstraction.

The example above tells us that we cannot make the obvious restriction on F: if $e = e'$ is an equation of the abstract type, then $F(M\{e\}) \approx F(M\{e'\})$. Instead we must build in the notion that contexts have restricted access to abstract types. This leads to the following definition (cf. definitions 1 and 2):

Definition 4. *1. Suppose F is a functor $D \longrightarrow C$, and $f, g : X \longrightarrow Y$ is a parallel pair of morphisms in D, then $f \sim_F g$ if and only if for all (A, B) in Obs, and for all $\alpha : iA \longrightarrow X$ and $\beta : Y \longrightarrow iB$, we have $F(\beta \cdot f \cdot \alpha) = F(\beta \cdot g \cdot \alpha)$.*

2. If, in addition, C, D and F are cartesian closed, then $\approx_F$ is the largest congruence on D contained in $\sim_F$ (i.e. $f \approx_F g$ if and only if for all U,V in $ob(D)$, $([U, f] \times V) \sim_f ([U, g] \times V)$).

The implementation is valid if and only if all equations $\boldsymbol{x} : \boldsymbol{X} \vdash e = e' : Y$ hold up to contextual equivalence, i.e., $M\{e\} \approx_F M\{e'\}$, or equivalently, the quotient $D \longrightarrow D/\approx_F$ factors through $D \longrightarrow D/\approx_E$. The only function of the equational theory is to restrict the notion of implementation. It has no effect on when two implementations are equivalent, and so now plays no further role in the story: it just, so to speak, comes along for the ride.

Definition 5. *Suppose $i : C \longrightarrow D$ is an injective-on-objects cartesian closed functor between cartesian-closed categories, and $\mathcal{E}$ is a set of parallel pairs of morphisms in D (which we view as the equations of an abstract type system), then an implementation F of $(D, \mathcal{E})$ in C is a cartesian closed functor $F : D \longrightarrow C$, such that $F \cdot i = id_C$, and for all $(f, g) \in \mathcal{E}$, $F(f) \approx_F F(g)$.*

We now observe some properties of $\approx_F$ that will help us to make precise what we mean by two implementations being equivalent.

Proposition 3. *Let $F : D \longrightarrow C$ be a cartesian closed functor such that $F \cdot i = id_C$. Then the following hold of $\approx_F$:*

1. *for all $f, g : X \longrightarrow Y$ in D, if $Ff \approx Fg$, then $f \approx_F g$.*
2. *for all $f, g : X \longrightarrow Y$ in C, if $f \approx g$, then $if \approx_F ig$.*
3. *for all $f, g : iX \longrightarrow iY$ in D, we have $f \approx_F g$ if and only if $Ff \approx Fg$.*
4. *for all $f, g : iA \longrightarrow iB$ in D, where $(A, B) \epsilon Obs$, we have $f \approx_F g$ if and only if $Ff = Fg$.*

Proof. 1. Suppose $Ff \approx Fg$. Then, given $\alpha : iA \longrightarrow [U, X] \times V$ and $\beta : [U, Y] \times V \longrightarrow iB$, we have $F(\beta \cdot ([U, f] \times V) \cdot \alpha) = F\beta \cdot ([FU, Ff] \times FV) \cdot F\alpha = F\beta \cdot ([FU, Fg] \times FV) \cdot F\alpha = F(\beta \cdot ([U, g] \times V) \cdot \alpha)$

2. Suppose $f \approx g$. Then, given $\alpha : iA \longrightarrow [U, iX] \times V$ and $\beta : [U, iY] \times V \longrightarrow iB$, we have $F(\beta \cdot ([U, if] \times V) \cdot \alpha) = F\beta \cdot ([FU, Fif] \times FV) \cdot F\alpha = F\beta \cdot ([FU, f] \times FV) \cdot F\alpha = F\beta \cdot ([FU, g] \times FV) \cdot F\alpha = F\beta \cdot ([FU, Fig] \times FV) \cdot F\alpha = F(\beta \cdot ([U, ig] \times V) \cdot \alpha)$

3. Suppose $f \approx_F g$. Then, given $\alpha : A \longrightarrow [U, X] \times V$ and $\beta : [U, Y] \times V \longrightarrow B$, we have $\beta \cdot ([U, Ff] \times V) \cdot \alpha = Fi\beta \cdot ([FiU, Ff] \times FiV) \cdot Fi\alpha = F(i\beta \cdot ([iU, f] \times iV) \cdot i\alpha) = F(i\beta \cdot ([iU, g] \times iV) \cdot i\alpha) = Fi\beta \cdot ([FiU, Fg] \times FiV) \cdot Fi\alpha = \beta \cdot ([U, Fg] \times V) \cdot \alpha$. The converse has already been shown.

4. $Ff \approx Fg$ if and only if $Ff = Fg$ for observable types.

Armed with these observations about the interaction of the definitions of $\approx$ and $\approx_F$, we can now establish relationships between a pair of interpretations F and G, where we understand an interpretation to be defined to be a functor $F : D \longrightarrow C$ such that $F \cdot i = id_C$.

Proposition 4. *Let $F, G : D \longrightarrow C$ be two interpretations. Then, the following are equivalent:*

1. *F and G induce the same equivalence: $\approx_F = \approx_G$*
2. *F and G induce the same equivalence on observable operations: $\approx_F |_{iObs} = \approx_G |_{iObs}$.*
3. *the interpretations of operations between pre-existing types are contextually equivalent: for all $f : iX \longrightarrow iY$, we have $Ff \approx Gf$.*
4. *the interpretations of observable operations are contextually equivalent (in fact, equal): for all $f : iA \longrightarrow iB$ where $(A, B) \epsilon Obs$, we have $Ff \approx Gf$.*

Proof. $1 \Rightarrow 2$ and $3 \Rightarrow 4$ are immediate.

$2 \Rightarrow 1$. Suppose $f \approx_F g : X \longrightarrow Y$, and we have $\alpha : iA \longrightarrow [U, X] \times V$ and $\beta : [U, Y] \times V \longrightarrow iB$. Then $F(\beta \cdot ([U, f] \times V) \cdot \alpha) = F(\beta \cdot ([U, g] \times V) \cdot \alpha)$. But these are between observable types, hence by Proposition 3(4), we have $\beta \cdot ([U, f] \times V) \cdot \alpha \approx_F \beta \cdot ([U, g] \times V) \cdot \alpha$. So $\beta \cdot ([U, f] \times V) \cdot \alpha \approx_G \beta \cdot ([U, g] \times V) \cdot \alpha$; and hence by Proposition 3(4) again, we have $G(\beta \cdot ([U, f] \times V) \cdot \alpha) = G(\beta \cdot ([U, g] \times V) \cdot \alpha)$.

$1 \Rightarrow 3$. Given $f : iX \longrightarrow iY$, we have $Ff = FiFf$, so by Proposition 3(1), we have $f \approx_F iFf$. So $f \approx_G iFf$, and hence by Proposition 3(3), we have $Gf \approx GiFf = Ff$.

$4 \Rightarrow 2$. Given $f, g : iA \longrightarrow iB$ where $(A, B) \epsilon Obs$, suppose $f \approx_F g$. Then $Ff = Fg$. But by hypothesis, $Ff = Gf$, and $Fg = Gg$, so $Gf = Gg$. Hence $f \approx_G g$.

Any of the above conditions would be a reasonable definition of equivalence of implementation. So the proposition conveniently justifies the following.

Definition 6. *Two interpretations $F, G : D \longrightarrow C$ are* equivalent *if they satisfy the equivalent conditions of Proposition 4.*

Example 3.

1. Suppose F and G correspond to two implementations of stack, in both of which the underlying type of stack is the product of an unbounded array of integers (giving the stack), and a non-negative integer (giving the top pointer). In F, pop is implemented by decrementing the top pointer, and leaving the stack unchanged. In G, the top pointer is decremented, and the cell popped is set to zero in the array. F and G are otherwise identical. Then F and G are equivalent implementations.
2. It is not necessary for the underlying types to be the same in the two implementations, if F is the array implementation of stack mentioned above, and G is a list implementation, then F and G will be equivalent exactly when they agree about top(empty) and pop(empty).

4 Logical Relations

In [8], Ma and Reynolds began by analysing the classical theory of logical relations for simple types. They focussed on the structure needed to prove the fundamental theorem. For binary relations, this amounts to a cartesian closed category B equipped with a cartesian closed functor $p = (p_0, p_1) : B \longrightarrow C \times C$, and that cartesian closed functor is typically a fibration [3]. In their theory, B is constructed concretely as a category of relations, and so comes with a standard diagonal $\Delta : C \longrightarrow B$. This is also a cartesian closed functor.

For instance, consider C being the category Set. Then an object of B consists of a pair (X, Y) of sets together with a subset R of $X \times Y$. The diagonal $\Delta : C \longrightarrow B$ is indeed the diagonal in the usual sense, taking a set X to the subset $\{(x, x) | x \in X\}$ of $X \times X$.

The significance of the fact that B has and all the functors preserve cartesian closed structure, together with the fact that $p = (p_0, p_1) : B \longrightarrow C \times C$ is a

fibration, is that logical relations can be defined on base types, and then extended inductively on the syntax of type expressions as usual. See [3] for a careful analysis of this. Technically, we do not need this much structure for soundness, but having it gives a stronger completeness result and a better link in to logics for reasoning about the type systems. So we take this structure as a characterisation of logical relations, but since our categories are more abstract than those of Ma and Reynolds, we require an abstract characterisation of the diagonal. Moreover, we would like the diagonal to characterise contextual equivalence, not equality.

In the standard relational setting, when $R \subseteq X_0 \times X_1$ and $S \subseteq Y_0 \times Y_1$, a map in B from R to S is exactly a pair of maps $f_0 : X_0 \longrightarrow Y_0$ and $f_1 : X_1 \longrightarrow Y_1$, such that $f_0 \times f_1$ maps R into S, i.e. a pair of maps which respects the relations R and S. In our more abstract setting it is not necessarily true that a map in B is given by a pair of maps in C, and so we shall regard a pair of maps (f_0, f_1) as respecting the "relations" R and S exactly when there is a map $f : R \longrightarrow S$ such that $p_0(f) = f_0$ and $p_1(f) = f_1$. This enables us to pick out the essential property we need of the diagonal functor: that if two maps preserve diagonals, then they are contextually equivalent.

Definition 7. *Let $p = (p_0, p_1) : B \longrightarrow C \times C$ be a cartesian closed functor, and let $\mathcal{D} : C \longrightarrow B$ be a cartesian closed functor such that $p \cdot \mathcal{D} = \Delta : C \longrightarrow C \times C$. Then we say $\mathcal{D}$ is* diagonal *if, for all $f : \mathcal{D}X \longrightarrow \mathcal{D}Y$, it follows that $p_0 f \approx p_1 f$.*

For a formal definition and exposition of the notion of fibration, one may see Claudio Hermida's thesis [3], half of which is about the relationship between logical relations and fibrations. The idea is that a fibration with structure, in particular cartesian closed structure, provides a category theoretic equivalent to having a predicate logic with which one can build logical relations. Formally, the definition is as follows.

Definition 8. *A functor $p : \mathcal{P} \longrightarrow \mathcal{C}$ is called a* fibration *if for every object X of $\mathcal{P}$ and every map of the form $g : A \longrightarrow pX$ in $\mathcal{C}$, there is an arrow $g^*_X : g^*(X) \longrightarrow X$ in $\mathcal{P}$ such that $p(g^*(X)) = A$ and $p(g^{*_X}) = g$, and the following universal property is satisfied:*

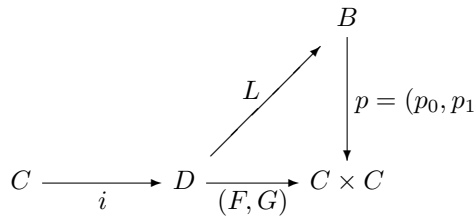
for every arrow $f : Z \longrightarrow X$ and every arrow $h : pZ \longrightarrow p(g^*(X))$ such that $g \cdot h = p(f)$, there exists a unique arrow $k : Z \longrightarrow g^*(X)$ such that $p(k) = h$ and $g^*_X \cdot k = f$.

Example 4. We recast our major example of binary relations as follows. $\mathcal{C}$ is $\mathsf{Set} \times \mathsf{Set}$. The objects of $\mathcal{P}$ are binary predicates (formally, a pair of sets (X_0, X_1), together with a predicate $R(x_0, x_1)$ on them). A morphism from $(X_0, X_1, R(x_0, x_1))$ to $(Y_0, Y_1, S(y_0, y_1))$ is a pair of functions (f_0, f_1), such that

$$\models \forall x_0, x_1.\ R(x_0, x_1) \Rightarrow S(f_0(x_0), f_1(x_1))$$

If $X = (X_0, X_1, R(x_0, x_1))$, and $g = (g_0 : Z_0 \longrightarrow X_0, g_1 : Z_1 \longrightarrow X_1)$, then $g^*(X) = (Z_0, Z_1, R(g_0(z_0), g_1(z_1)))$.

Definition 9. *F and G are* linked by a logical relation *if there is a cartesian closed category B and a cartesian closed fibration $p : B \longrightarrow C \times C$ together with a cartesian closed functor $L : D \longrightarrow B$ lying over (F, G) (i.e. $p_0 \cdot L = F$ and $p_1 \cdot L = G$) such that $\mathcal{D} = L \cdot i$ is diagonal.*



We can now immediately prove the soundness of this form of logical relation for proving equivalence of implementations.

Proposition 5. *If F and G are two implementations which are linked by a logical relation, L, then F and G are equivalent.*

Completeness involves a variant of Jung and Tiuryn's Kripke logical relations of varying arity.

5 Jung-Tiuryn's Logical Relations of Varying Arity

In this section, we outline how we prove the completeness part of our main result. Given equivalent interpretations $F, G : D \longrightarrow C$, we seek a cartesian closed fibration $p : B \longrightarrow C \times C$ and a cartesian closed functor from D to B. Our construction is motivated by Jung and Tiuryn's construction of Kripke logical relations of varying arity [5], though it corresponds to a variant and not their precise construction. (It is possible to prove the result we want in a way that corresponds precisely to theirs. The construction is, however, less compact and less obviously generalisable.)

First, Jung and Tiuryn's choice of name is questionable. Their paper is concerned with definability in the simple type hierarchy. Definability is a unary predicate, and there is a precise sense in which the relations they define are, despite appearances to the contrary, unary relations. We try to explain this viewpoint below giving first an informal motivational account, and then an account of the machinery which can be used to make the presentation rigorous. Our informal account will confuse terms and their semantics.

Jung and Tiuryn's work can be reconstructed as follows. We want to provide an invariance property of the lambda definable functions. We hope this can be expressed by a logical relation. In other words, we hope that the set of λ-definable functions is a unary logical relation. Unfortunately in the standard picture, it is not. This is because in the standard theory we can only talk about the elements of the sets which are carriers (closed terms), and there are plenty of functions which send definable elements to definable elements, but are not themselves definable. One way of looking at Jung and Tiuryn's achievement is

that they found a clever way of talking about open terms. This allows us to take our operation and apply it to the definable open term "x". If the result is definable, by e say, then the operation must have been definable by $\lambda x.e$.

In Jung and Tiuryn's account the open terms appear semantically, and stratified by context. That is to say we are given a signature of (typed) variables which may appear in the operation, and the term (say of type D) appears as a function from the set of environments for that signature to D. Instead of viewing this as a function, Jung and Tiuryn view it as a tuple whose components are indexed by environments, and hence as an element of a relation on D whose arity is the set of environments. In order that a property of an open term remains true when we enlarge the environment, Jung and Tiuryn use a Kripke structure in which the worlds are the possible signatures. This means that from their perspective, what they have is a relation which is always on the set D, but whose arity varies as we move between worlds. However, another way of looking at the same structure is that the relation is always unary, but that the set on which is a relation is the set of terms definable at that world, and hence varies from world to world.

The standard way to obtain logical relations is to take a semantics in Set, and to use subsets. A theme of this paper is that this can equally well be regarded as using the standard logic of Set. This can be varied by using other categories and other logics. The technical tool we use for doing this is to interpret a logic as a fibration with structure. If the logic admits $\top$, $\wedge$, $\rightarrow$ and $\forall$, as well as substitution, then it yields a category of logical relations. The observation that these operations are all that is required to define product and exponential of logical relations should make that immediately credible, even it does not furnish a proof. These two approaches meet in the standard logic of predicates interpreted as subobjects.

In standard Kripke logical relations for the simple type hierarchy, we are given an indexing category W. The types of the hierarchy are interpreted as constant functors in the presheaf topos $[W^{op}, Set]$, so $(\Delta d)w = d$. Kripke logical relations are derived from the natural logic of the topos, that is predicates correspond to subfunctors (a subfunctor of F is a functor G such that at each world w, $Gw \subseteq Fw$). For connected W, the functor $\Delta : Set \longrightarrow [W^{op}, Set]$ is a full and faithful cartesian closed functor. This means that the simple type hierarchy on d in Set is sent by Δ to the simple type hierarchy on Δd in $[W^{op}, Set]$. So we can either regard this construction as the form of logical relations obtained from a standard logic for a non-standard set in non-standard set theory, or we can regard it as a non-standard logic for a standard set in standard set theory. The Jung-Tiuryn construction resolves this in favour of the second.

In the Jung-Tiuryn construction, W is assumed to be a concrete category, i.e., a category of sets and functions. So they have a functor $J : W \longrightarrow Set$, which they assume to be faithful (although never using that), and essentially what they do is replace the functor $\Delta : Set \longrightarrow [W^{op}, Set]$ above by the functor $\tilde{J} : \mathsf{Set} \longrightarrow [W^{op}, Set]$, sending d to the functor $\mathsf{Set}[J-, d]$.

We make two minor modifications of this. First we start with a functor, $F : D \longrightarrow C$. We turn it around to obtain the functor $\tilde{F} : C \longrightarrow [D^{op}, Set]$

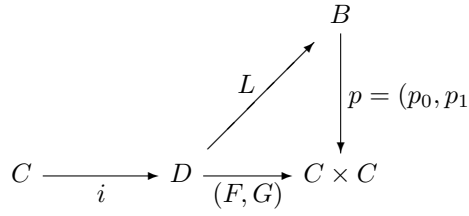
sending X to the functor $C(F-,X) : D^{op} \longrightarrow Set$. We get unary logical relations by considering the subobject fibration over $[D^{op}, Set]$ (the standard logic there), and we can take the pullback of that cartesian closed fibration along $\tilde{F}$ to obtain a cartesian closed fibration over C (a non-standard logic). This involves some delicacy. Although the functor F is cartesian closed, it does not follow that $\tilde{F}$ is cartesian closed. But $\tilde{F}$ does preserve finite products, which is sufficient for the results we seek.

Next we take a slightly unexpected binary version of this. It would be normal, given a functor $K : C \longrightarrow [D^{op}, Set]$ to consider the pullback of a fibration over $[D^{op}, Set] \times [D^{op}, Set]$ along $K \times K$. But that is not what we propose. We have functors $\tilde{F}, \tilde{G} : C \longrightarrow [D^{op}, Set]$ and we shall take the pullback along $\tilde{F} \times \tilde{G}$. We do obtain the result we seek, but this is a most unusual construction. For instance, there seems no way to lift $\tilde{F} \times \tilde{G}$ to a functor from the binary subobject fibration $Sub_2(C)$ over $C \times C$ to the binary subobject fibration $Sub_2([D^{op}, Set])$ over $[D^{op}, Set] \times [D^{op}, Set]$, but that is a fundamental construction of fibrations, simply not one we need here.

Finally, one needs a little caution. Ideally, we would like the construction of our cartesian closed fibration $p : B \longrightarrow C \times C$ to be independent of the choice of F and G, with F and G only being required in order to construct $L : D \longrightarrow B$. But we cannot see how to do that: that is a particular sense in which we see the relationship with Jung and Tiuryn's work, as they have an extra parameter $J : X \longrightarrow Set$ in their work too, and it was that extra parameter that inspired us.

6 Completeness of Logical Relations

In this section, we consider the completeness half of the result we seek. So, given a pair $F, G : D \longrightarrow C$ of equivalent interpretations, we seek a cartesian closed fibration $p : B \longrightarrow C \times C$ and a cartesian closed functor $L : D \longrightarrow B$ lying over (F, G) such that the composite $L \cdot i$ is diagonal.



As mentioned in the previous section, given an arbitrary functor $F : D \longrightarrow C$, we consider the functor $\tilde{H} : C \longrightarrow [D^{op}, Set]$ that sends an object X of C to the functor $C(F-,X) : D^{op} \longrightarrow Set$. It follows from the Yoneda lemma that we have

Proposition 6. *For any $F : D \longrightarrow C$, if C has finite products, then the functor $\tilde{F} : C \longrightarrow [D^{op}, Set]$ preserves finite products.*

Also, as a standard example of a fibration [3], we have

Example 5. Let $\mathcal{C}$ be any category with finite limits. Let $Sub_2(\mathcal{C})$ have objects $(X, Y, R \to X \times Y)$ consisting of a pair (X, Y) of objects of $\mathcal{C}$ together with a subobject R of $X \times Y$, and with the evident arrows. Then the forgetful functor $Sub_2(\mathcal{C}) \longrightarrow \mathcal{C} \times \mathcal{C}$ is a fibration. If $\mathcal{C}$ is cartesian closed, the fibration is a cartesian closed fibration. It is the standard fibration of binary predicates for $\mathcal{C}$.

Proposition 7. *Given categories $\mathcal{C}$ and $\mathcal{C}'$ with finite products, a finite product preserving functor $K : \mathcal{C}' \longrightarrow \mathcal{C}$, and a finite product preserving fibration $p : \mathcal{P} \longrightarrow \mathcal{C}$, then the pullback $K^*(\mathcal{P})$ of p along K gives a finite product preserving fibration $K^*(\mathcal{P})$ over $\mathcal{C}'$.*

A proof of this result appears in Hermida's thesis [3], but it can also be verified by direct calculation.

Corollary 1. *Given $F, G : D \longrightarrow C$ where C has finite products, the pullback of $Sub_2[D^{op}, Set]$ along $\tilde{F} \times \tilde{G}$ gives a finite product preserving fibration $(\tilde{F} \times \tilde{G})^*(Sub_2[D^{op}, Set])$ over $C \times C$.*

In fact, the fibration $(\tilde{F} \times \tilde{G})^*(Sub_2[D^{op}, Set])$ is the one we want, so we shall duly denote it by B. This is essentially a *sconing* of the ordinary binary subobject fibration on the presheaf category $[D^{op}, Set]$. It is easy to prove by direct calculation, using some pullbacks in the presheaf category, that we have

Theorem 1. *Given $F, G : D \longrightarrow C$, supposing C is cartesian closed, then $B = (\tilde{F} \times \tilde{G})^*(Sub_2[D^{op}, Set])$ is a cartesian closed fibration over $C \times C$.*

Proof. The only point here that requires checking is that B is cartesian closed and that the functor to $C \times C$ preserves the cartesian closed structure. This follows from a careful analysis of what the objects of B are. Given an object (X, X') of $C \times C$, an object of B over (X, X') is a subfunctor of $C(F-, X) \times C(G-, X')$. Using cartesian closedness of $Sub_2([D^{op}, Set]$ over $[D^{op}, Set] \times [D^{op}, Set]$, given subfunctors of $C(F-, X) \times C(G-, X')$ and $C(F-, Y) \times C(G-, Y')$, one can construct a subfunctor of the exponential $[C(F-, X) \times C(G-, X'), C(F-, Y) \times C(G-, Y')]$. Finally, taking the pullback of that subfunctor along the canonical comparison map from $C(F-, [X, Y]) \times C(F-, [X', Y'])$ to $[C(F-, X) \times C(G-, X'), C(F-, Y) \times C(G-, Y')]$ gives the exponential in B that we seek.

This is a specific example of a general construction [3]. Now we have the data we require in order to state and prove our main result.

Theorem 2. *If F and G are equivalent interpretations, they are linked by a logical relation.*

Proof. Define B as in Theorem 1. It remains to define $L : D \longrightarrow B$ over (F, G), prove it is cartesian closed, and prove that $L \cdot i$ is diagonal. So for an object d of D, define Ld to be the subfunctor of $C(F-, Fd) \times C(G-, Gd)$ given by

$$Ld(c) = \{(g : Fc \longrightarrow Fd, h : Gc \longrightarrow Gd) \mid \exists f : c \longrightarrow d.\ Ff \approx g \wedge Gf \approx h\}$$

It is routine to verify that L preserves finite products; it takes considerably greater but still routine effort to verify that it preserves cartesian closed structure. It follows directly from the definition and the fact that F and G are equivalent that $L \cdot i$ is diagonal.

Combining this completeness result with the soundness result of the previous section, we have a combined soundness and completeness result as follows.

Theorem 3. *Two interpretations F and G are equivalent if and only if they are linked by a logical relation.*

References

1. Alimohamed, A.: A Characterisation of Lambda Definability in Categorical Models of Implicit Polymorphism. Theoretical Computer Science **146** (1995) 5–23.
2. Fiore, M., Plotkin, G.D.: An axiomatisation of computationally adequate domain-theoretic models of FPC. Proc LICS **94**. IEEE Press (1994) 92–102.
3. Hermida, C.A.: Fibrations, Logical Predicates and Indeterminates. Ph.D. thesis. Edinburgh (1993) available as ECS-LFCS-**93-277**.
4. Honsell, F., Sannella, D.T.: Pre-logical relations. Proc. CSL **99**. LNCS **1683**. Flum and Rodriguez-Artalejo (eds.) Springer (1999) 546–562.
5. Jung, A., Tiuryn, J.: A new characterisation of lambda definability. Proc. TLCA. LNCS **664**. Bezem and Groote (eds.) Springer (1993) 245–257.
6. Kinoshita, Y., O'Hearn, P., Power, A.J., Takeyama, M., Tennent, R.D.: An Axiomatic Approach to Binary Logical Relations with Applications to Data Refinement. Proc TACS **97**. LNCS **1281**. Abadi and Ito (eds.) Springer (1997) 191–212.
7. Lambek, J., Scott, P.J.: Introduction to higher order categorical logic. Cambridge studies in advanced mathematics. Cambridge University Press (1986)
8. Ma, Q., Reynolds, J.C.: Types, abstraction and parametric polymorphism **2**. Math. Found. of Prog. Lang. Sem. Lecture Notes in Computer Science. Springer (1991).
9. Mitchell, J.:Type systems for programming languages. Handbook of Theoretical Computer Science em B. van Leeuwen (ed). North-Holland (1990) 365–458.
10. Mitchell, J.: Foundations for programming languages. Foundations of Computing Series. MIT Press (1996).
11. Moggi, E.: Computational Lambda-calculus and Monads. Proc LICS **89**. IEEE Press (1989) 14–23.
12. Plotkin, G.D., Power, A.J., Sannella, D.T., Tennent, R.D.: Lax logical relations. (to appear in ICALP 2000).
13. Reynolds, J.C.: Types, abstraction, and parametric polymorphism. Information Processing **83** Mason (ed) North Holland (1983) 513–523.
14. Tennent, R.D.: Correctness of data representations in ALGOL-like languages. In: A Classical Mind, Essays in Honour of C.A.R. Hoare, A.W. Roscoe (ed.) Prentice-Hall (1994) 405–417.

Elementary Choiceless Constructive Analysis

Peter M. Schuster

Ludwig-Maximilians-Universität München
Mathematisches Institut, Theresienstraße 39, 80333 München, Germany
pschust@rz.mathematik.uni-muenchen.de

Abstract. Existential statements seem to admit a constructive proof without countable choice only if the object to be constructed is uniquely determined, or is intended as an approximate solution of the problem in question. This conjecture is substantiated by re-examining some basic tools of mathematical analysis from a choice-free constructive point of view, starting from Dedekind cuts as an appropriate notion of real numbers. As a complement, the question whether densely defined continuous functions do approximate intermediate values is reduced to connectivity properties of the corresponding domains.

Key Words and Phrases. Constructive Mathematics, Countable Choice, Unique Existence, Approximate Analysis, Intermediate Values, Connectedness

2000 MSC. Primary 03F60; Secondary 03E25, 26A15, 26E40, 54D05, 54E35

1 Unique Existence and Countable Choice

Following Beeson ([2], page 25, footnote 16),

> Bridges has observed that in general, existence theorems seem to be constructive when the object whose existence is in question is unique. Otherwise put, non-constructive theorems always involve non-uniqueness. ... In practice, whenever a theorem is known to be non-constructive, the solution whose existence is non-constructive is also non-unique. Conversely, the difficulty in constructivizing certain problems ... seems to be intimately related to the fact that the solutions are not known to be (locally) unique.

The purpose of this article is to reconsider Bridges's conjecture by concentrating on constructive proofs which, in addition, require as little countable choice as possible. Making therefore explicit every subsequent invocation of choice principles, we proceed in the context of Bishop's constructive mathematics ([3]; see also [4,12], and, for a general overview, [2,35]). In its today's liberal interpretation advocated by Bridges, Richman, and others (see, e.g., [27]), this is, roughly speaking, mathematics carried out by intuitionistic logic[1], and thus simultane-

[1] Anybody questioning whether *ex falso quodlibet* (EFQ) is really used in constructive practice might take into account that, even within minimal logic, EFQ is equivalent with $(P \vee Q) \wedge \neg Q \to P$.

P. Clote and H. Schwichtenberg (Eds.): CSL 2000, LNCS 1862, pp. 512–526, 2000.

ously generalises classical, (constructive) recursive, and intuitionistic mathematics. According to Bridges [6], constructive mathematics in Bishop's sense is also suitable as a framework for computational analysis of whatever kind.

Here is our first and guarded claim:

Locally unique existence is closely related with choice-free construction.

One might be able to by-pass choice also in many—if not all—of the rather frequent situations where a particular problem possibly has several solutions:

- uniqueness can be forced by gathering beforehand all the solutions which belong to each other in a natural way;
- the question of local (non)uniqueness can be avoided from the outset by concentrating on approximate instead of exact solutions.

Postponing examples that illustrate either point, let us notice that the former supports the claim we made above whereas the latter leads to the second half of our thesis:

Constructions of approximate solutions do presumably not require choice.

We now briefly explain the role of choice principles within constructive mathematics. Although the axiom of choice in its full form can never be transferred to any constructive framework for entailing the certainly nonconstructive principle of *tertium non datur*[2], two special cases have yet frequently been invoked by most constructive mathematicians:

Countable Choice (CC). *Given a sequence $(A_n)_{n\in\mathbb{N}}$ of nonempty sets, there is a sequence $(a_n)_{n\in\mathbb{N}}$ such that $a_n \in A_n$ for every $n \in \mathbb{N}$.*

Dependent Choice (DC). *If A is a nonempty set and $S \subset A \times A$ such that for every $a \in A$ there is some $a' \in A$ with $(a, a') \in S$, then for each $a_0 \in A$ there is a sequence $(a_n)_{n\in\mathbb{N}}$ in A, beginning with a_0, such that $(a_n, a_{n+1}) \in S$ for all $n \in \mathbb{N}$.*

[2] Diaconescu [17] and Goodman-Myhill [18] have proven this within intuitionistic topos theory and constructive set theory, respectively, supposing that—as usual in constructive mathematics, too, where equality is a defined relation—any (choice) function f is extensional, that is, $x = y \Rightarrow f(x) = f(y)$ for all x, y.

Let us briefly summarise this argument, following [1]. Given any proposition P, consider the sets

$$A = \{x \in \{0,1\} \colon P \vee x = 0\}, \quad B = \{x \in \{0,1\} \colon P \vee x = 1\},$$

and note that both A and B are nonempty for $0 \in A$ and $1 \in B$ in any case. If $f : \{A, B\} \to A \cup B$ is a choice function for these data, that is, $a = f(A) \in A$ and $b = f(B) \in B$, then both $P \vee a = 0$ and $P \vee b = 1$ or, equivalently, either P or $a = 0 \wedge b = 1$; whence we only have to derive $\neg P$ from $a = 0 \wedge b = 1$. To this end, assume P; then $A = \{0, 1\} = B$ and thus $a = b$ by the extensionality of f, a contradiction. See also footnote 3.

Whereas countable choice is nothing else but the countable version of full choice, dependent choice is slightly stronger, but appears to be indispensable whereever infinite sequences are 'constructed' step-by-step. The justification of either principle common to nearly all schools of constructive mathematics is based on the Brouwer-Heyting-Kolmogorov interpretation (BHK) of any $\forall n \exists a$ statement, which says that some algorithm $n \rightsquigarrow a$ has to be in the background. Moreover, since integers are said to be given as such, not requiring any extra presentation, every (of course, deterministic) algorithm fed with integer inputs defines a function with integer arguments—in other words, a sequence[3].

In spite of this good argument for accepting countable and dependent choice, there are just as good reasons for rejecting these putatively constructive choice principles. The idea to refrain from their use in constructive mathematics was brought up by Ruitenburg [31] and subsequently put forward by Richman [28]; further substantial steps in the direction of a choice-free constructive mathematics have been made since. We refer to [29] for an overview from Richman's own standpoint, including several case studies which clearly show the virtues of doing constructive mathematics without countable choice.

One of the main counterarguments is that the use of choice for solving parametrised equations seems to hinder the solutions from depending continuously upon the parameters: choice might enable us to switch between the different branches of the solution, thus producing points of discontinuity; even 'choosing' a fixed branch would become rather useless as soon as we cross some branching

[3] Nothing else but BHK stands also behind the derivation of the choice principle particular to Martin-Löf's intuitionistic type theory (ITT); see pages 50–52 of [25]. This circumstance has even been noted *expressis verbis* in the middle of page 50 ibid.: "The same idea [the justification of choice by means of BHK] can be put into symbols, getting a formal proof [of the choice principle in ITT] ...". Because the domains of choice functions in ITT are allowed to be 'sets' without any apparent restriction, it is necessary to stress that in ITT every set has to come along with special rules for, e.g., introduction and elimination (op.cit., page 24), just as $\mathbb{N}$ is formed from the initial element 0 by the successor operation and embodies the principle of induction. If only in this sense, choice in ITT could be related with CC, notwithstanding the fact that, unlike $\mathbb{N}$, sets in ITT are in general neither denumerable nor equipped with a decidable equality.

In view of the fact that ITT is a definitely constructive theory, it is reasonable to ask why the provability of the choice principle belonging to ITT does not conflict with what we have recalled in footnote 2, namely, that the full axiom of choice entails the law of excluded middle. The answer to this question is neither that, unlike sometimes suspected, choice functions in ITT need not be extensional (in general, they need), nor that A and B do not fit the demands of a set in ITT (in spite of their somewhat pathological character, they do). What hinders us from applying the ITT version of choice to this situation is rather that, in ITT, something like $\{A, B\}$ cannot be equipped with the *extensional* equality according to which A and B are identified precisely when they possess the same elements—an essential assumption in footnote 2. Such set formation, however, becomes possible as soon as ITT is enriched by extensional power sets or effective quotients, constructors which indeed infect ITT with classical logic; see Maietti [20] and Maietti-Valentini [21].

point. If, on the other hand, the solutions are uniquely determined by the parameters, such discontinuity phenomena are impossible; moreover, solutions are then functions in the parameters and thus continuously dependent on the latter. In constructive mathematics, namely, solutions have to be algorithms with the parameters as inputs, and functions with discontinuities cannot be defined at all[4].

Why exact solutions might require more choice than approximate ones should become clearer when choice is viewed as a uniformisation principle, transforming any $\forall\exists$ statement into the corresponding $\exists\forall$ statement. To this end, let us consider some continuous real-valued function f on the unit interval $I = [0, 1]$, and suppose that we are given the conclusion of the approximate intermediate value theorem, that is,

$$(1) \qquad \forall \varepsilon > 0\, \exists x \in I\colon |f(x)| < \varepsilon.$$

From this fact we could extract—but presumably *only* by countable choice—some sequence (x_n) in I with $|f(x_n)| < 1/n$ for all $n \geqslant 1$. Disregarding that, for localising some cluster point in I of the sequence (x_n), in most cases we needed the sequential compactness of I which is a rather nonconstructive principle[5], any such cluster point would constitute a witness for $\exists x \in I\colon f(x) = 0$.

However, by simply writing this conclusion of the exact intermediate value theorem in the less usual but equivalent form

$$(2) \qquad \exists x \in I\, \forall \varepsilon > 0\colon |f(x)| < \varepsilon\,,$$

one realises which role choice plays in this context: assuming what shall be argued for in section 4, namely, that statements like (1) seem to allow choice-free proofs, countable choice proves to be the price one has to pay for the extra step towards (2) unless something else helps, for instance, unique existence.

2 Completeness of Real Numbers

There are fewer Cauchy reals than Dedekind reals in the absence of countable choice[6], which is also very likely to be indispensable for proving any completeness property of Cauchy reals[7]. This situation can already be related with

[4] In intuitionistic and constructive recursive mathematics, functions on continuous domains such as intervals can even be proven to be pointwise continuous; compare [12], chapter 6.

[5] See, however, [8,9] for investigations of the constructive content of this principle.

[6] According to [12], pages 138–140, the rationals are embedded as globally constant functions into the choice-free intuitionistic model of the reals that consists of all the continuous functions on some (classical) topological space. Because statements about reals are local properties in this model, every continuous function can be approximated arbitrarily closely by rationals, whereas any limit of a sequence of rationals has to be locally constant.

[7] Unless Cauchy sequences are equipped with Cauchy moduli in the sense of [31] and [35], chapter 5. As Richman noticed, a Cauchy sequence of Cauchy reals in

(non)uniqueness phenomena: although the limit of some convergent sequence is uniquely determined up to equality, no canonical 'choice' can be made of any rational Cauchy sequence representing a given real number.

Dedekind reals, on the other hand, admit almost by definition (and without choice) the following well-known version of order completeness, which seems to be stronger than sequential completeness in the absence of countable choice.

Least-Upper-Bound Principle (LUB). *A nonempty set S of real numbers that is bounded above possesses a supremum provided that*
(*) *given reals $\alpha < \beta$, either $s \leqslant \beta$ for all $s \in S$ or $\alpha < r$ for some $r \in S$.*

A *supremum* of S is a real number σ such that

- $\triangleright$ $s \leqslant \sigma$ for all $s \in S$;
- $\triangleright$ if ρ is a real number with $\rho < \sigma$, then $\rho < r$ for some $r \in S$.

Note that each supremum of S is a least upper bound in the usual sense; in particular, it is uniquely determined, and we may denote it by $\sup S$. Moreover, condition (*) is even *necessary* for the existence of a supremum, and it suffices to check (*) for any ordered pair $\alpha < \beta$ of *rational* numbers. LUB is, of course, equivalent with the analogous statement about infima, or greatest lower bounds.

By a *Dedekind real* we understand a located Dedekind cut in the rationals, that is, a pair (L, U) of disjoint nonempty open subsets of $\mathbb{Q}$ such that either $p \in L$ or $q \in U$ for all $p, q \in \mathbb{Q}$ with $p < q$. The strict partial order of Dedekind reals is given by $(L, U) < (L', U')$ if and only if $L' \setminus L$ or, equivalently, $U \setminus U'$ is nonempty; the weak partial order is given by $(L, U) \leqslant (L', U')$ if and only if $L \subset L'$ or, equivalently, $U' \subset U$. Inequality $\neq$ is, of course, the disjunction of $<$ and $>$; equality $=$ as the conjunction of $\leqslant$ and $\geqslant$ is nothing else but the usual equality relation between pairs of sets. Referring to section 4 of [34] for further details, we write $\mathbb{R}$ for the set of Dedekind reals.

In $\mathbb{R}$, however, as in every constructive model of the reals, one ought to be careful with the use of negation: although it is readily seen that $x \leqslant y$ coincides with $\neg(x > y)$ and that, consequently, $x = y$ can be identified with $\neg(x \neq y)$, it is quite obvious that $x < y$ and $x \neq y$ are constructively stronger than $\neg(x \geqslant y)$ and $\neg(x = y)$, respectively. Let us point out that we shall often employ

$$(\dagger) \qquad x < y \Rightarrow x < z \vee z < y\,,$$

an axiom scheme easily justified for Dedekind reals which has proven to be a good substitute for the nonconstructive law of dichotomy $z \leqslant 0 \vee z \geqslant 0$.

Let $\mathfrak{R}$ be an archimedean ordered Heyting field, that is, a model of Bridges's [6] set of axioms *minus* LUB. These axioms (of course, together with LUB) embody all the properties of real numbers that are commonly accepted in constructive mathematics, as there is (†); needless to say, $\mathbb{R}$ is a perfect model of

the modulated context is nothing else but a doubly indexed sequence of rational numbers, a notion lacking one of the main features of Cauchy sequences in general, namely, that they can be utilised for completing *arbitrary* metric spaces. Compare, however, footnote 20.

those axioms. In particular, any such $\mathfrak{R}$ contains $\mathbb{Q}$ as a dense subfield in the sense that for all $x, y \in \mathfrak{R}$ with $x < y$ there is some $q \in \mathbb{Q}$ such that $x < q < y$.

The following theorem has been pointed out to us by Fred Richman.

Theorem 1. *LUB is valid for $\mathfrak{R}$ if and only if there is an order preserving mapping $j : \mathbb{R} \to \mathfrak{R}$ that operates as identity on $\mathbb{Q}$.*

Proof. If $\sigma = (L, U) \in \mathbb{R}$, then $L \subset \mathbb{Q}$—considered as a subset of $\mathfrak{R}$—satisfies condition (*). Provided LUB for $\mathfrak{R}$, set $j(\sigma) = \sup L$; it is routine to verify that j is order preserving, and that $j|\mathbb{Q}$ is the identity. Conversely, if $S \subset \mathfrak{R}$ fulfills the hypotheses of LUB, then $\sigma_S = (L_S, U_S) \in \mathbb{R}$ with L_S, U_S as the open kernels of

$$\{p \in \mathbb{Q} \mid \exists s \in S\colon p < s\}, \quad \{q \in \mathbb{Q} \mid \forall s \in S\colon s \leqslant q\},$$

respectively; moreover, $j(\sigma_S)$ is the supremum of S. □

Let us underline that we shall utilise the notions of Cauchy sequence and of convergence of sequences in $\mathfrak{R}$ only when LUB is valid for $\mathfrak{R}$, in order to have $|x| = \max\{x, -x\}$ for any $x \in \mathfrak{R}$. Indeed, (†) implies (*) for any S of the form $\{x_1, \ldots, x_n\}$; whence $\max\{x_1, \ldots, x_n\}$ and $\min\{x_1, \ldots, x_n\}$ exist in presence of LUB for all $x_1, \ldots, x_n \in \mathfrak{R}$.

The derivation of sequential completeness from LUB given by Bridges [6] can easily be rendered choice-free; we nevertheless prove this fact in the way particular to Dedekind reals, following page 132 of [34]:

Corollary 1. *If LUB obtains for $\mathfrak{R}$ then every Cauchy sequence in $\mathfrak{R}$ converges.*

Proof. Each Cauchy sequence (x_n) in $\mathfrak{R}$ determines a Dedekind real $\xi = (L_\xi, U_\xi)$: let L_ξ, U_ξ be the open kernels of

$$\{p \in \mathbb{Q} \mid \exists N\, \forall n \geqslant N\colon p < x_n\}, \quad \{q \in \mathbb{Q} \mid \exists N\, \forall n \geqslant N\colon x_n < q\},$$

respectively. Moreover, (x_n) converges to $j(\xi)$. □

Corollary 2. *Dedekind reals satisfy LUB and are sequentially complete.* □

In particular, $\mathbb{R}$ is a model of the whole set of Bridges's axioms [6], including LUB.

All these results can equally be applied to the formal reals presented by Negri-Soravia [26], real numbers developed within the (constructive and predicative) formal topology due to Sambin [32,33]. Very roughly speaking, a formal real α consists of pairs of rationals $p < q$ such that the open intervals $]p, q[$ form a neighbourhood base of α. It is easily checked that the natural bijection between formal and Dedekind reals is an order isomorphism (compare [26], sections 5 and 9); hence we get LUB for formal reals, too[8]. Every model akin to formal

[8] In order to avoid impredicativity, one might interpret the set S in LUB as a family indexed by a sufficiently neat set; confer Proposition 6.3 of [26], which is LUB *without* the hypothesis (*) and thus providing a *weak* formal real as supremum. Weak formal reals, however, are little satisfying because they can hardly have all the features of their strong counterparts. For a choice-free proof of the sequential completeness of formal reals not employing their weak version, see [26], Theorem 8.6.

reals therefore appears to be as suitable as Dedekind reals for any choice-free approach[9]; however, if only for the sake of a uniform presentation, we have chosen to concentrate on the latter: from now on, we understand by a real number always a Dedekind real.

3 Exact Intermediate Value Theorems

Given real numbers $a < b$, set $[a, b] = \{x \in \mathbb{R} : a \leqslant x \leqslant b\}$. Let us recall the

Intermediate Value Theorem (IVT). *If $f : [a, b] \to \mathbb{R}$ is a continuous*[10] *function with $f(a) \leqslant 0 \leqslant f(b)$, then $f(x) = 0$ for some $x \in [a, b]$.*

Being almost folklore that IVT is nonconstructive unless some hypotheses are added[11], it is noteworthy that the well-known example of a continuous function f that 'balks' at IVT is *nondecreasing* in the sense that $x < y \Rightarrow f(x) \leqslant f(y)$ for all x, y (see, e.g., [35], 6.1.2). As Helmut Schwichtenberg pointed out to us, the classical interval halving argument still applies to functions mapping rationals to rationals, such as polynomials with rational coefficients, for which even countable choice is unnecessary because the rationals are totally ordered.

Theorem 2. *IVT is valid for every pointwise continuous $f : [a, b] \to \mathbb{R}$ with $f(\mathbb{Q}) \subset \mathbb{Q}$, provided that $a, b \in \mathbb{Q}$ or $f(a) < 0 < f(b)$.*

Proof. We may assume $a, b \in \mathbb{Q}$: if, e.g., $f(a) < 0$, then $f(a') < 0$ for some $a' \in \mathbb{Q} \cap]a, b[$. Hence there is a *uniquely determined* nested sequence of nonempty intervals $I_n \subset [a, b]$ with rational endpoints, beginning with $I_0 = [a, b]$, such that I_{n+1} is the left (right) half of I_n whenever $f(c_n) \geqslant 0$ (< 0) for $c_n \in \mathbb{Q}$ as the midpoint of I_n. In particular, (I_n) shrinks to a real number x with $f(x) = 0$. □

The putatively most general constructive version of IVT is the one for functions f that are *locally nonconstant*, which is to say that whenever $a \leqslant x < y \leqslant b$ then $f(z) \neq 0$ for some $z \in [x, y]$. Including the extensive class of (nonconstant) real-analytic functions, this extra condition rules out those 'balking' functions which *are* locally constant somewhere, but still allows functions to possess multiple zeros: consider, for instance, $f(x) = x^2 - c$ for $c \geqslant 0$. Accordingly, dependent choice appears to be necessary for proving this form of IVT by approximate interval halving (see, e.g., [35], 6.1.5).

[9] Notwithstanding the fact that a rather general choice principle is a built-in tool of formal topology, as of ITT; see footnote 3.

[10] Albeit following Bishop's supposition that any continuous function on $[a, b]$ is uniformly continuous ([4], page 38), we shall subsequently make explicit when pointwise continuity suffices. What Bishop simply postulated fails in constructive recursive mathematics but is derivable in intuitionistic and, of course, in classical mathematics; compare [12], chapter 6.

[11] IVT implies the law of dichotomy (DICH) for real numbers which in turn entails the 'lesser limited principle of omniscience' (LLPO), a statement provably false in constructive recursive mathematics; see [12], pages 53, 56 for details. In fact, IVT, DICH, and LLPO are equivalent, if only by CC.

The same proof method was used beforehand (e.g., on page 40 of [4]) for the IVT for *strictly increasing* functions, that are functions f such that $x < y \Rightarrow f(x) < f(y)$ for all x, y [12]. Since strictly increasing functions have at most one zero, the reader might already expect that countable choice is dispensable for this particular type of functions. Indeed, a choice-free proof by interval tesselating has recently been noted by Richman ([30], Theorem 4); let us nevertheless provide a rather order-theoretic proof.

Theorem 3. *IVT obtains for every strictly increasing pointwise continuous f.*

Proof. Having proven that $S = \{x \in [a, b] : f(x) \leqslant 0\}$ possesses a supremum, it is routine to verify that $\sup S \in [a, b]$ and $f(\sup S) = 0$. To check the hypothesis (*) of LUB, note first that if $\alpha < \beta$ then either $\alpha < a \vee b < \beta$, in which case we are done, or $a < \beta \wedge \alpha < b$ and thus $\max\{a, \alpha\} < \min\{b, \beta\}$; in particular, we may assume $\alpha, \beta \in [a, b]$. For then $f(\alpha) < f(\beta)$, either $f(\alpha) < 0$ or $f(\beta) > 0$; in the former case, $f(\alpha') < 0$ and thus $\alpha' \in S$ for some $\alpha' \in]\alpha, b]$, whereas in the latter case β is easily seen to be an upper bound of S. □

It is tempting to generalise the choice-free approach to *strictly injective* functions, by which we mean those f with $x \neq y \Rightarrow f(x) \neq f(y)$ for all x, y [13]; of course, any strictly injective nondecreasing function is strictly increasing. Since strictly injective functions are locally nonconstant, one could derive from the IVT for locally nonconstant functions (that with choice) that any strictly injective continuous f is either strictly increasing or else strictly decreasing, depending on whether $f(a) < f(b)$ or $f(a) > f(b)$, respectively.

There is another positive monotonicity property which at first glance seems to be suitable for some IVT without choice, namely, $f(x) < f(y) \Rightarrow x < y$ for all x, y; this property was named *antidecreasing* by Mandelkern [23] [14]. Any nondecreasing function f is antidecreasing provided that f is also *strongly extensional*, which is to say that $f(x) \neq f(y) \Rightarrow x \neq y$ for all x, y. Since, however, *all* pointwise continuous functions are strongly extensional [15], we cannot expect to prove IVT for antidecreasing functions.

4 Some Approximate Analysis

Throughout this section, let (M, d) be a metric space, and $a, b \in M$. Recall the

Approximate Intermediate Value Theorem (aIVT). *If $f : M \to \mathbb{R}$ is a continuous function with $f(a) \leqslant 0 \leqslant f(b)$, then for every $\varepsilon > 0$ there is some $x \in M$ with $|f(x)| < \varepsilon$.*

[12] Note also that, in both sources just referred to, one does not really have to suppose $f(a) < 0 < f(b)$.

[13] Note that this property is constructively stronger (and thus more appropriate) than its contrapositive $f(x) = f(y) \Rightarrow x = y$ for all x, y.

[14] Bridges-Mahalanobis [10,11], who simply called it *increasing*, have demonstrated that this property allows to detect the possible discontinuities of a given function, and to extend the domain of any partial function to all points where left-hand and right-hand limit exist and coincide.

[15] See also Ishihara [19] for the relation between continuity and strong extensionality.

We will subsequently investigate conditions on M and f that ensure the validity of aIVT under these particular hypotheses. Let us stress that we shall frequently but tacitly invoke the principle (†) pointed out in section 2.

Although there might well be no path in M that connects a and b, some x as in aIVT can at least be found on any such path: as we will see later on, aIVT obtains for intervals. Moreover, the hypothesis $0 \in [f(a), f(b)]$ can be replaced by $0 \in \langle f(a), f(b)\rangle$ where $\langle u, v\rangle = \{u + tv : t \in [0, 1]\}$ is the *convex hull* of any $u, v \in \mathbb{R}$[16]: in fact, given $\varepsilon > 0$, either $|f(a) - f(b)| > 0$, i.e., $f(a) \neq f(b)$ and thus, e.g., $f(a) < f(b)$, in which case $\langle f(a), f(b)\rangle = [f(a), f(b)]$ and we are done by aIVT; or $|f(a) - f(b)| < \varepsilon$, in which case $\max\{|f(a)|, |f(b)|\} < \varepsilon$ anyway.

In [22] and [35], page 381, a topological space is called *connected* if $U \cap V$ is nonempty whenever $U \cup V$ is a nontrivial open covering. The following fact in case $M = [0, 1]$ was noted by Mandelkern [23], who in turn ascribes it to Ray Mines and Fred Richman.

Theorem 4. *aIVT is valid for every pointwise continuous f provided that M is connected.*

Proof. Since $M = U \cup V$ for the open sets

$$U = \{x \in M : f(x) < \varepsilon\}, \quad V = \{x \in M : f(x) > -\varepsilon\},$$

there is, by hypothesis, some $x \in U \cap V$; in other words, $|f(x)| < \varepsilon$. □

The connectedness of any interval, however, seems to rely on dependent choice (confer [22], Theorem 2 and [35], 6.1.3); observe that this existential statement is essentially lacking uniqueness. As Helmut Schwichtenberg noted[17], the direct proof of aIVT for intervals can nevertheless be rendered choice-free; it suffices to substitute the approximate interval halving argument still used on page 40 of [4] by interval tesselating. We will now slightly generalise this method.

Let us call M *almost connected* whenever, for all nonempty subsets R, S of M, if $M = R \cup S$ then for every $\varepsilon > 0$ there are $r \in R$, $s \in S$ with $d(r, s) < \varepsilon$.

Proposition 1. *Every connected metric space M is almost connected.*

Proof. Given $\emptyset \neq R, S \subset M$ with $M = R \cup S$, the open sets

$$U = \{x \in M : d(x, r) < \varepsilon/2 \text{ for some } r \in R\},$$
$$V = \{x \in M : d(x, s) < \varepsilon/2 \text{ for some } s \in S\}$$

cover M. By hypothesis, there is $x \in U \cap V$; hence $d(x, r) < \varepsilon/2$ and $d(x, s) < \varepsilon/2$ for some $r \in R$ and $s \in S$, respectively, and thus $d(r, s) < \varepsilon$. □

Theorem 5. *aIVT obtains for every uniformly continuous f provided that M is almost connected.*

[16] Compare [5] and [24], 10.12, 16.4; there is a constructive difference between $[u, v]$ and $\langle u, v\rangle$ for possibly incomparable $u, v \in \mathbb{R}$; what we have in general is $\langle u, v\rangle \subset [u, v]$ for $u \leqslant v$ and $\langle u, v\rangle = [u, v]$ for $u < v$.

[17] Yet unpublished lecture notes.

Proof. Given $\varepsilon > 0$, f is uniformly $(\varepsilon/2)$-δ continuous for some $\delta > 0$. Since $M = R \cup S$ with

$$R = \{x \in M : f(x) < \varepsilon/2\}, \quad S = \{x \in M : f(x) > -\varepsilon/2\},$$

there are, by hypothesis, $r \in R$ and $s \in S$ with $d(r, s) < \delta$; hence $|f(r)| < \varepsilon$ and $|f(s)| < \varepsilon$. □

Let us furthermore call M *nearly connected* if for all $a, b \in M$ and $\varepsilon > 0$ there are $c_0, \ldots, c_n \in M$ such that $c_0 = a$, $c_n = b$, and $d(c_k, c_{k-1}) < \varepsilon$ for $k = 1, \ldots, n$; in other words, a and b can be connected by a chain of finitely many points in M with mesh $< \varepsilon$.

Proposition 2. *Every nearly connected metric space is almost connected.*

Proof. For any $\emptyset \neq R, S \subset M$ with $M = R \cup S$, pick $a \in R$, $b \in S$. Given $\varepsilon > 0$, there are $c_0, \ldots, c_n \in M$ with $a = c_0$, $b = c_n$, and $d(c_k, c_{k-1}) < \varepsilon$ for all k. Since $c_0 \in R$, $c_n \in S$, and $c_k \in R$ or $c_k \in S$ for all k, one can find, by hypothesis, some k such that $r = c_{k-1} \in R$ and $s = c_k \in S$. By construction, $d(r, s) < \varepsilon$. □

Proposition 3. *If M is a dense subset of some pathwise connected space X, then M is nearly connected.*

Proof. Let $\gamma : [0, 1] \to X$ be a path connecting $a, b \in M$. Given $\varepsilon > 0$, γ is uniformly $(\varepsilon/3)$-δ continuous for some $\delta > 0$. Construct a tesselation

$$0 = t_0 < \ldots < t_n = 1$$

of $[0, 1]$ with mesh $< \delta$, pick $c_k \in M$ with $d(c_k, \gamma(t_k)) < \varepsilon/3$ for $k = 1, \ldots, n-1$, and set $c_0 = a$, $c_n = b$. Then the chain $c_0, \ldots, c_n$ in M has mesh $< \varepsilon$. □

We do not dare to ask under which circumstances the completion of some nearly connected metric space is pathwise connected.

Lemma 1. *Each almost connected metric space M is nearly connected provided that $M = M_1 \cup \ldots \cup M_m$ for nearly connected subspaces $M_1, \ldots, M_m$ of M.*

Proof. Given $a, b \in M$ and $\varepsilon > 0$, we may assume $a \in M_1$ and $b \in M_m$. Since $M = R \cup S$ for $R = M_1 \cup \ldots \cup M_{m-1}$ and $S = M_m$, there are, by hypothesis, $r \in R$ and $s \in S$ with $d(r, s) < \varepsilon$. By induction, one can find a chain with mesh $< \varepsilon$ consisting of finitely many elements of R connecting a and r. □

Recall that a metric space M is *totally bounded* if, for every $\varepsilon > 0$, M can be covered by finitely many open balls of radius ε with centres in M.

Proposition 4. *Each almost connected metric space M is nearly connected provided that M is totally bounded, or consists of finitely many path components.*

Proof. In view of Proposition 3 we may assume that M is the union of finitely many nearly connected subspaces; Lemma 1 concludes the proof. □

Corollary 3. *aIVT obtains for every uniformly continuous f provided that M is a dense subset of some pathwise connected metric space.* □

It is noteworthy that we have not yet proven aIVT for functions on compact intervals—one cannot expect to show constructively that these rather simple spaces are pointwise connected, let alone convex[18]; we need some approximate convexity notion instead.

In section 10 of [24], a metric space M is called *nearly convex* whenever, for any $x, y \in M$ and $\lambda, \mu > 0$, if $d(x,y) < \lambda + \mu$ then $d(x,z) < \lambda$ and $d(y,z) < \mu$ for some $z \in M$; furthermore, any subset M of $\mathbb{R}$ is called *paraconvex* whenever, for any $x, y \in M$ and $z \in \mathbb{R}$, if $x \leqslant z \leqslant y$ then $z \in M$. According to [24], 10.13, paraconvex subsets of $\mathbb{R}$ are nearly convex.

Proposition 5. *If M is a dense subset of an interval, then M is nearly convex.*

Proof. No matter whether real numbers a, b can be compared with each other, it is obvious that $a \leqslant x, y \leqslant b$ and $x \leqslant z \leqslant y$ entail $a \leqslant z \leqslant b$. In particular, intervals are paraconvex and thus nearly convex; the latter property is transmitted to dense subsets. □

Proposition 6. *Every nearly convex metric space M is nearly connected.*

Proof. Given $a, b \in M$ and $\varepsilon > 0$, set $c_0 = a$, $c_n = b$, and $\rho_0 = d(c_0, c_n)$. Either $\rho_0 < \varepsilon$ and we are done, or $\rho_0 > \varepsilon/2$, in which case, by hypothesis, there is $c_1 \in M$ with $d(c_0, c_1) < \varepsilon$ and $d(c_n, c_1) < \rho_0 - \varepsilon/2 = \rho_1$. By repeating this process with c_0, ρ_0 substituted by c_1, ρ_1, we get—after a finite number of steps—some chain $c_0, \ldots, c_n$ in M with mesh $< \varepsilon$. □

Corollary 4. *aIVT is valid for every uniformly continuous f whenever M is a dense subset of an interval.* □

Needless to say, approximate results for intervals can easily be carried over to convex subsets of normed spaces. In view of Proposition 6, one might suspect that the notions 'nearly connected' and 'nearly convex' coincide with each other, but by removing the hypothenuse from some rectangular triangle one gets a nearly connected space that is not nearly convex. However, following the proof of [24], Theorem 10.7, where 'connected' is supposed instead of 'almost connected', we realise that for subsets of the line all approximate notions are equivalent.

Proposition 7. *Every almost connected subset M of $\mathbb{R}$ is nearly convex.*

Proof. Given $x, y \in M$ and $\lambda, \mu > 0$ such that $|x - y| < \lambda + \mu$, either $|x - y| < \min\{\lambda, \mu\}$, in which case we are done, or $x \neq y$; hence we may assume $x < y$. For $\varepsilon > 0$ with $y - x < (\lambda - \varepsilon) + (\mu - \varepsilon)$ and $\varepsilon < \min\{\lambda, \mu\}$, set

$$R = M \cap]-\infty, x + \lambda - \varepsilon[\,, \quad S = M \cap]y - \mu + \varepsilon, +\infty[\,.$$

[18] Although it remains to find a Brouwerian counterexample for that compact intervals cannot be pointwise connected, there is one for that they cannot be convex: following [24], 10.10, the assumption that $[-|x|, +|x|]$ is convex for given $x \in \mathbb{R}$ would enable a decision whether $x \geqslant 0$ or $x \leqslant 0$ (this observation answers also the question on page 69 of [24]). In intuitionistic mathematics, Waaldijk has proven that at least the real numbers in $[0,1]$ admitting a ternary expansion form a pathwise connected space; note that we cannot expect every real number to possess a ternary expansion ([36], 0.2.1, 0.2.2, 2.1.6). Open intervals, on the other hand, are easily seen to be convex.

Since $M = R \cup S$ because of $y - \mu + \varepsilon < x + \lambda - \varepsilon$, we get, by hypothesis, $|r - s| < \varepsilon$ for some $r \in R$ and $s \in S$; hence $z \in M \cap]y - \mu, x + \lambda[$ and thus $|x - z| < \lambda$ and $|y - z| < \mu$ for $z = r$ or $z = s$. □

Corollary 5. *For each subset M of $\mathbb{R}$, the following items are equivalent.*

(i) *M is nearly convex.*
(ii) *M is nearly connected.*
(iii) *M is almost connected.* □

Again, we hesitate to conjecture something like that the closure of a nearly convex subset of $\mathbb{R}$ is an interval. Note that, finally, $\mathbb{Q}$ is nearly convex but not connected: $\sqrt{2}$ cuts $\mathbb{Q}$ into two open halves.

5 Concluding Remarks

Although the notion of a strictly increasing function is—at least for practical purposes—not as restrictive as it seems to be, the corresponding IVT (without choice) is less satisfying than that for locally nonconstant functions (with choice): unlike the latter[19], the former can hardly be extended to functions on arbitrary normed spaces, let alone metric spaces. The case is just as for Cauchy sequences versus Dedekind cuts in the rationals[20]: general Euclidean space lacks the additional structure of the linear continuum that is given by order. In other words, there literally is 'more choice' in higher dimensions, for instance, of geometrical directions, and these possible choices can hardly be by-passed unless one abolishes or neglects them by forcing the solutions to be unique or by concentrating on approximate solutions, respectively. By the way, the (choice-free) approximate form of IVT is from the very outset not restricted to functions of a single variable.

The phenomenon just mentioned can already be observed in the case of two dimensions. When dealing with complex numbers, namely, one is often inclined to use polar coordinates $z = |z| \exp(i \arg z)$, and every complex number $z \neq 0$ can indeed be equipped with an argument $\arg z$ by means of the (choice-free) IVT for strictly increasing functions. However, no argument can be constructed for arbitrary—possibly vanishing—complex numbers z: otherwise every real number (even those close to 0) would admit some sign ± 1, a property which is nothing else but the nonconstructive dichotomy principle for real numbers[21]. The situation is as for the classical form of IVT that would, of course, suffice for a general

[19] See Bridges's generalisation to functions on arbitrary normed spaces [5].

[20] As Richman noticed, one might well proceed from the rationals to the reals by a method such as Dedekind cuts by which one cannot complete arbitrary metric spaces—to speak about a metric requires a notion of real numbers given in advance, unless one completes the underlying space and the range of the metric simultaneously.

[21] See Bridges-Dediu [7] for a direct proof of the facts that polar coordinates imply LLPO, and that polar coordinates with the additional property $e^{i \arg z} \neq 1 \Rightarrow z \neq 0$ even entail the stronger 'limited principle of omniscience' (LPO).

polar decomposition: solutions of possibly locally constant equations cannot be isolated constructively.

How to calculate nevertheless square roots of complex numbers, a purpose for which complex numbers were designed in their origin? Of course, one easily derives from $i^2 = -1$ the well-known formulas giving both square roots of a complex number $z = x + iy$, provided that $x \neq 0$ or $y \neq 0$. Problems arise, just as for polar coordinates, only in the neighbourhood of 0: in order to localise a single root of a complex number z close to 0, one has to choose between the two possible roots as soon as $z \neq 0$ turns out to be the case—in other words, choice has to enter the stage.

Some rather specific (and classically valid) countable choice principle presented in [14] happens, however, to suffice for constructing some square root of an arbitrary complex number z without making use of the alternative $z \neq 0 \vee z = 0$, namely,

Weak Countable Choice (WCC). *Given a sequence $(A_n)_{n\in\mathbb{N}}$ of nonempty sets at most one of which is not a singleton, there is some choice sequence $a_n \in A_n$ $(n \in \mathbb{N})$.*

Roughly speaking, WCC enables the extraction of a choice sequence provided that there is at most one 'true choice' between two possibly different objects, no matter at which stage this occurs, if at all.

More generally, an entirely choice-free constructive proof of the fundamental theorem of algebra has been given by Richman in [28]; his construction produces as output the whole 'multiset' of roots of any input polynomial, from which one can extract a single element by means of WCC; see, again, [14] for the latter method. Note that Richman could only get completely rid of countable choice by gathering all roots of some polynomial together and thus forcing the solution in question to be uniquely determined, whereas one needs some choice—namely, WCC—as soon as one gives up this uniqueness demand in order to get a single root.

After all, one might still suspect that choice-free constructive mathematics cannot deal with infinite-dimensional spaces. That, on the contrary, Hilbert spaces can well be handled without countable choice has been shown recently in [13,14,15,16], where one can found proofs of unique existential statements such as

- the Riesz representation theorem ([15], Theorem 3), and
- the fact that each point in a strictly convex normed space has a closest point to any complete located subset ([13], Theorem 6),

as well as various results of approximative character.

Let us end with a quotation from the same source with which we have started our considerations (Beeson [2], page 25, footnote 16):

> He [Bridges] wonders why this is. Logicians, is there a meta-theorem to explain it [that constructive proofs are related with unique existence]?

We would like to extend Beeson's question by asking for reasons why constructions without countable choice seem to require locally unique existence (or some turning to 'approximate mathematics'), reasons somewhat deeper than the mere indications we have given in this article.

Acknowledgements

Preliminary versions of this paper were designed during visits at the Technische Universität Wien, Institut für Analysis und Technische Mathematik, and at the Università degli Studi di Padova, Dipartimento di Matematica Pura ed Applicata; the author is indebted to Rudolf Taschner and Giovanni Sambin for the respective invitations. He also wishes to express his gratitude to Fred Richman and Helmut Schwichtenberg for numerous communications, as well as to Venanzio Capretta, Milly Maietti, Giovanni Sambin, and, in particular, to Giovanni Curi for patiently explaining parts of Martin-Löf's theory. The detailed suggestions by one of the anonymous referees have helped a lot to bring this article into its final form.

References

1. Bell, J.L., Zorn's lemma and complete boolean algebras in intuitionistic type theories. J. Symb. Logic **62** (1997), 1265-1279
2. Beeson, M.J., Foundations of Constructive Mathematics. Ergebn. Math. Grenzgeb., 3. Folge, Bd. 6. Springer, Berlin and Heidelberg, 1985
3. Bishop, E., Foundations of Constructive Analysis. McGraw-Hill, New York, 1967
4. Bishop, E., Bridges, D., Constructive Analysis. Grundl. math. Wiss., Bd. 279. Springer, Berlin and Heidelberg, 1985
5. Bridges, D.S., A general constructive intermediate value theorem. Z. math. Logik Grundl. Math. **53** (1989), 433–453
6. Bridges, D.S., Constructive mathematics: a foundation for computable analysis. Theoret. Comp. Science **219** (1999), 95–109
7. Bridges, D.S., Dediu, L.S., Paradise lost, or Paradise regained? Europ. Assoc. Theor. Comp. Science Bull. **63** (1997), 141 – 145
8. Bridges, D., Ishihara, H., Schuster, P., Sequential compactness in constructive analysis. Österr. Akad. Wiss., Math.-Nat. Kl., Sitzungsber. II **208** (1999)
9. Bridges, D., Ishihara, H., Schuster, P., Compactness and continuity revisited. Forthcoming.
10. Bridges, D., Mahalanobis, A., Constructive continuity of increasing functions. Forthcoming.
11. Bridges, D., Mahalanobis, A., Increasing, nondecreasing, and virtually continuous functions. Forthcoming.
12. Bridges, D., Richman, F., Varieties of Constructive Mathematics. Cambridge University Press, 1987
13. Bridges, D., Richman, F., Schuster, P., Linear independence without choice. Ann. Pure Appl. Logic **101** (2000), 95–102
14. Bridges, D., Richman, F., Schuster, P., A weak countable choice principle. Proc. Amer. Math. Soc., to appear

15. Bridges, D., Richman, F., Schuster, P., Adjoints, absolute values and polar decomposition. J. Oper. Theory, to appear
16. Bridges, D., Richman, F., Schuster, P., Trace-class operators. Forthcoming.
17. Diaconescu, R., Axiom of choice and complementation. Proc. Amer. Math. Soc. **51** (1975), 176–178
18. Goodman, N.D., Myhill, J., Choice implies excluded middle. Z. math. Logik Grundl. Math. **23** (1978), 461
19. Ishihara, H., Continuity and nondiscontinuity in constructive mathematics. J. Symb. Logic **56** (1991), 1349–1354
20. Maietti, M.E., About effective quotients in constructive type theory. In: T. Altenkirch, W. Naraschewski, B. Reus (eds.), Types '98. Types for Proofs and Programs. Proc. 1998 Irsee Meeting. Springer Lecture Notes in Computer Science **1657** (1999)
21. Maietti, M.E., Valentini, S., Can you add power-sets to Martin-Löf intuitionistic set theory? Math. Logic Quart. **45** (1999), 521–532
22. Mandelkern, M., Connectivity of an interval. Proc. Amer. Math. Soc. **54** (1976), 170–172
23. Mandelkern, M., Continuity of monotone functions. Pac. J. Math. **99** (1982), 413–418
24. Mandelkern, M., Constructive continuity. Mem. Amer. Math. Soc. **42** (1983), no. 277
25. Martin-Löf, P., Intuitionistic Type Theory (Notes by G. Sambin of a series of lectures given in Padua, June 1980). Bibliopolis, Napoli, 1984
26. Negri, S., Soravia, D., The continuum as a formal space. Arch. Math. Logic **38** (1999), 423–447
27. Richman, F., Intuitionism as generalization. Philos. Math. **5** (1990), 124–128
28. Richman, F., The fundamental theorem of algebra: a constructive treatment without choice. Pac. J. Math., to appear
29. Richman, F., Constructive mathematics without choice. In: U. Berger, H. Osswald, P. Schuster (eds.), Reuniting the Antipodes. Constructive and Nonstandard Views of the Continuum. Proc. 1999 Venice Symposion, forthcoming.
30. Richman, F., Pointwise differentiability. Forthcoming in the same volume as [29].
31. Ruitenburg, W.B.G., Constructing roots of polynomials over the complex numbers. In: A. M. Cohen (ed.), Computational Aspects of Lie Group Representations and Related Topics. Proc. 1990 Computer Algebra Seminar, Centrum voor Wiskunde en Informatica (CWI), Amsterdam. CWI Tract **84** (1991), 107–128
32. Sambin, G., Intuitionistic formal spaces—a first communication. In: D. Skordev (ed.), Mathematical Logic and its Applications. Proc. 1986 Bulgaria Conference. Plenum, New York and London, 1987
33. Sambin, G., Formal topology and domains. In: Proc. Workshop Domains IV, Informatik-Berichte 99-01, Universität-Gesamthochschule Siegen, 1999
34. Schuster, P.M., A constructive look at generalised Cauchy reals. Math. Logic Quart. **46** (2000), 125–134
35. Troelstra, A.S., van Dalen, D., Constructivism in Mathematics. An Introduction. Two volumes. North Holland, Amsterdam, 1988
36. Waaldijk, F., Modern Intuitionistic Topology. Proefschrift, Katholieke Universiteit Nijmegen, 1996

On the Logic of the Standard Proof Predicate

Rostislav E. Yavorsky *

Steklov Mathematical Institute RAS
Gubkina 8, GSP-1, Moscow, RUSSIA, 117966
tel. (095) 938-3752, fax (095) 135-0555
rey@mi.ras.ru

Abstract. In [2] S. Artemov introduced the logic of proofs $\mathcal{LP}$ describing provability in an arbitrary system. In this paper we present the logic $\mathcal{LPM}$ of the standard multiple conclusion proof predicate in Peano Arithmetic with the negative introspection operation. We establish the completeness of $\mathcal{LPM}$ with respect to the intended arithmetical semantics. Two useful artificial semantics for $\mathcal{LPM}$ were also found. The first one is an extension of the usual boolean truth tables, whereas the second one deals with so-called protocolling extension of a theory. For both cases the completeness theorem has been established.
In the last section we consider first order version of the logic $\mathcal{LPM}$. Arithmetical completeness of this logic is established too.

Keywords: logic of proofs, semantics, protocolling extensions of theories.

1 Introduction

The study of explicit provability logics or logics of proofs was initiated by S. N. Artemov in [1]. In [2] he presented the operational logic of proofs $\mathcal{LP}$ with the atoms "t is a proof of F" and established that every theorem of the modal logic $S4$ admits a reading in $\mathcal{LP}$ as the statement about explicit provability. This completed the effort by Kolmogorov [9] and Gödel [6,7] to provide a Brouwer – Heyting – Kolmogorov style classical provability semantics for intuitionistic logic.

In addition, it turned out that $\mathcal{LP}$ subsumes the λ-calculus, modal λ-calculus and combinatory logic. Recently, it was shown in [3] that this new approach to studying provability is useful for design of advanced system of proof verification.

A semantics for the logic $\mathcal{LP}$ was studied in [10]. In this paper we present two different semantics for the logic of proofs with the monotonicity axiom. In section 2 the logic $\mathcal{LPM}$ is introduced. The so called basic semantics is described in section 3. In section 4 we prove that $\mathcal{LPM}$ is arithmetically complete for the multiple conclusion version of the standard Gödel proof predicate. In section 5

* The work is partially supported by the Russian Foundation for Basic Research, grants 98-01-00249, 99-01-01282, INTAS grant 97-1259, and grant DAAH04-96-1-0341, by DARPA under program LPE, project 34145.

P. Clote and H. Schwichtenberg (Eds.): CSL 2000, LNCS 1862, pp. 527–541, 2000.

we introduce the notion of protocolling extension of a theory. The completeness of $\mathcal{LPM}$ with respect to interpretations into such an extensions is established. In section 6 we consider the first order logic $\mathcal{QLP}_{\mathcal{M}}$ and prove arithmetical completeness theorem for it.

2 Operational Logic $\mathcal{LPM}$ with the Monotonicity Axiom

The language of the logic $\mathcal{LPM}$ is an extension of the propositional language with infinite number of proof variables $p_1, p_2, \ldots$, symbols of operations $\cdot, +, !, ?_A$ and binary proof predicate $t:F$.

The definition of a proof term and a formula is given as follows:

1. propositional constants and variables $\top, \bot, S_0, S_1, \ldots$ are formulas (atoms);
2. if A, B are formulas then $\neg A$, $(A \wedge B)$, $(A \vee B)$, $(A \to B)$ are formulas too;
3. proof variables $p_1, p_2, \ldots$ are proof terms;
4. if s, t are proof terms then $(s \cdot t)$, $(s + t)$, $!t$ are proof terms too;
5. if t is a proof term and A is a formula then $?_A(t)$ is a proof term;
6. if A is a formula, t is a proof term then $(t:A)$ is a (quasiatomic) formula.

Thus, $\cdot$, $+$, $!$ are ordinary functional symbols (binary and unary correspondingly), while $?_A$ is a unary functional symbol with a formula as a parameter. In what follows $A(t)$ means that the proof term t occurs at least once as a subword in the formula A.

Definition 1 *The logic $\mathcal{LPM}$ is defined by the following axioms (over the propositional calculus in the extended language).*

Operational axioms:

1. $s:(A \to B) \to (t:A \to (s \cdot t):B)$ *(composition),*
2. $(t:A) \vee (s:A) \to (t + s):A$ *(non-deterministic choice),*
3. $t:A \to !t:(t:A)$ *(verification),*
4. $\neg\, t:A \to (?_A t):(\neg\, t:A)$ *(negative introspection).*

General axioms:

5. $t:A \to A$ *(reflexivity),*
6. $\neg(t_1:A_1(t_2) \wedge t_2:A_2(t_3) \wedge \ldots \wedge t_n:A_n(t_1))$ *(monotonicity).*

The rule of inference modus ponens $\dfrac{A, A \to B}{B}$.

This system is an extension of S. Artemov's logic $\mathcal{LP}$ (see [2]). We add the axiom of negative introspection and the monotonicity axiom (it appeared for the first time in [1]). The former is an operational version of the modal principle $\neg\Box A \to \Box\neg\Box A$ from the modal logic $S5$. It has very clear computational meaning: if some construction denoted by t is not a proof of a formula A this fact could be established effectively. The corresponding proof of non-correctness of t with respect to A is denoted by $?_A(t)$.

The monotonicity axiom characterizes a general property of considered interpretations, namely the fact that any proof of a formula A is more complex then the formula A itself. So, $s:A(t)$ implies $s \succ t$, where "$\succ$" denotes the corresponding irreflexive order. In the definition of $\mathcal{LPM}$ this property is expressed without additional symbol for the order relation.

3 The Basic Semantics for $\mathcal{LPM}$

Let us consider the language L^δ which includes the propositional language, and for an arbitrary finite set of formulas $\delta = \{\varphi_1, \ldots, \varphi_k\}$ and a formula φ the construction $[\delta]\varphi$ is a formula of L^δ. Here are some examples of L^δ-formulas:

$$[\varphi, \psi]\varphi, \quad [\psi, \neg\varphi]\varphi \vee [\varphi]\varphi, \quad [[\varphi]\varphi, \psi, \varphi]\psi \to [\psi]\varphi \text{ etc.}$$

Below, any finite set of L^δ-formulas is called a *configuration.*

Let $*$ be an evaluation of the propositional variables by the truth values from the set $\{0, 1\}$. We extend $*$ to all formulas of the language L^δ in the following way:

> $[\varphi_1, \ldots, \varphi_k]\psi$ is true iff all formulas $\varphi_1, \ldots, \varphi_k$ are true (i. e. the configuration is *correct*) and the formula ψ coincides with some formula φ_i from this set.

It is clear that the set of valid formulas of the language L^δ is decidable and could be axiomatized over the propositional calculus in the language L^δ by the following axioms:

D1. $[\varphi_1, \ldots, \varphi_k]\psi \leftrightarrow \varphi_1 \wedge \ldots \wedge \varphi_k$, if $\psi \in \{\varphi_1, \ldots, \varphi_k\}$;
D2. $\neg[\varphi_1, \ldots, \varphi_k]\psi$ otherwise.

We define the operations $\cdot$, $+$, $!$ and $?_\varphi$ on the set of all configurations such that for any configurations δ, δ_1, δ_2 and formulas φ, ψ of the language L^δ the following holds:

$(\cdot)$ if $(\varphi \to \psi) \in \delta_1$ and $\varphi \in \delta_2$ then $\psi \in (\delta_1 \cdot \delta_2)$;
$(+)$ if $\varphi \in \delta_1$ or $\varphi \in \delta_2$ then $\varphi \in (\delta_1 + \delta_2)$;
$(!)$ if $\varphi \in \delta$ then $[\delta]\varphi \in !(\delta)$;
$(?_\varphi)$ if $\varphi \notin \delta$ then $(\neg[\delta]\varphi) \in ?_\varphi(\delta)$.

We say that the operations are correct with respect to a given evaluation $*$ if when applied to correct configurations they return a correct configuration.

Note, that the operations defined in the minimal possible way (when the result is a minimal configuration which satisfies the conditions above) are correct for any evaluation $*$.

Definition 2 *A basic model for $\mathcal{LPM}$ is a triple $M = (*, Op, v)$ where*

$*$ *is a truth evaluation of propositional variables;*
Op *is a set of operations $\cdot$, $+$, $!$ and $?_\varphi$ which are correct with respect to the evaluation $*$;*
v *is a mapping of proof variables of the language $\mathcal{LPM}$ into the set of correct configurations of the language L^δ.*

The assignment v could be extended to the set of all formulas and proof terms of the language $\mathcal{LPM}$ in a natural way. We stipulate that v commutes with the boolean connectives and operations $\cdot$, $+$, $!$ and

$$v(S_i) \rightleftharpoons S_i, \qquad v(t{:}A) \rightleftharpoons [v(t)]v(A), \qquad v(?_A t) \rightleftharpoons ?_{v(A)}(v(t)).$$

It is clear that for every proof term t of the language $\mathcal{LPM}$ the configuration $v(t)$ is correct with respect to the evaluation $*$. A formula A is defined to be true in the described model if $v(A)^* = 1$.

Theorem 1 *Let A be any formula of the language $\mathcal{LPM}$. Then*

$$\mathcal{LPM} \vdash A \text{ iff } A \text{ is true in all basic models } (*, Op, v).$$

Proof. ($\Rightarrow$) For the operational axioms of $\mathcal{LPM}$ the statement follows immediately from the definition of the correct operations. As an example we check that for any basic model M one has

$$M \models s{:}(A \to B) \to (t{:}A \to (s \cdot t){:}B).$$

It means that $M \models [v(s)](v(A) \to v(B)) \to ([v(t)]v(A) \to [v(s \cdot t)]v(B))$. If $(v(A) \to v(B)) \notin v(s)$ or $v(A) \notin v(t)$ then the argument is trivial since one of the premises turns to be false. If $(v(A) \to v(B)) \in v(s)$ and $v(A) \in v(t)$ then by the definition of the operation $\cdot$ the configuration $v(s \cdot t)$ is correct in M and $v(B) \in v(s \cdot t)$. So, $M \models [v(s \cdot t)]v(B)$.

The other operational axioms are treated in a similar way.

The correctness of the monotonicity axiom follows from the following observation. If $M \models t_1 : A(t_2)$ then the length of the configuration $v(t_1)$ (we mean the number of symbols in its recording) is greater then the length of $v(t_2)$ since $v(t_2)$ occurs explicitly (as a part of the formula $v(A)$) in the configuration $v(t_1)$. So, in every sequence of formulas of the kind

$$t_1{:}A_1(t_2),\ t_2{:}A_2(t_3),\ \ldots,\ t_n{:}A_n(t_1)$$

one always can find such an element that $v(A_i) \notin v(t_i)$ and so $M \models \neg t_i{:}A_i$.

To complete this part of the proof one only need to check that if $M \models A \to B$ and $M \models A$ then $M \models B$.

($\Leftarrow$) Suppose that $\mathcal{LPM} \nvdash A$ and $t_1, \ldots, t_m$ is an exhaustive enumeration without repetitions of all terms which occur in A. Let us fix the set of fresh propositional variables $Q_1, \ldots, Q_m$. We define X_A to be the minimal set satisfying the following conditions.

1. Each atomic and each quasiatomic subformula of A belongs to X_A.
2. If a proof term $?_B t$ occurs in A and $t{:}B \in X_A$ then $?_B t{:}(\neg t{:}B) \in X_A$.
3. If a proof term $s \cdot t$ occurs in A and $s{:}(B \to C) \in X_A$, $t{:}B \in X_A$, then $(s \cdot t){:}C \in X_A$.
4. If a proof term $s + t$ occurs in A and $s : B \in X_A$ or $t : B \in X_A$, then $(s + t){:}B \in X_A$.
5. If a proof term $!t$ occurs in A and $t{:}B \in X_A$ then $!t{:}(t{:}B) \in X_A$.
6. For every atomic and quasiatomic formula from X_A its negation belongs to X_A too.
7. The following formulas $t_1{:}(Q_1 \vee \neg Q_1), \ldots, t_m{:}(Q_m \vee \neg Q_m)$ belong to X_A (this artificial condition is responsible for the injectivity of the constructed interpretation; it will be necessary for the definition of operations).

Note that the set X_A is finite since for every quasiatomic formula $t : B$ from X_A the proof term t occurs in A, and one can show by induction on the complexity of t that for any proof term the set of formulas $\{B_i \mid t{:}B_i \in X_A\}$ is finite.

Below, we will write "$\vdash$" instead of "$\mathcal{LPM} \vdash$".

A set of formulas $W \subset X_A$ is called to be *consistent,* if $\nvdash \bigwedge W \to \bot$. A consistent subset W of the set X_A is called to be *maximal,* if every extension of it with a formula from X_A is inconsistent. The following properties of the maximal consistent subsets easily follow from the definition.

Lemma 11 *Let W be some maximal consistent subset of X_A. Then*

1. *for every atomic or quasiatomic formula B from X_A one has either $B \in W$ or $\neg B \in W$;*
2. *for every subformula B of the formula A one has either $\vdash \bigwedge W \to B$ or $\vdash \bigwedge W \to \neg B$;*
3. *there exists maximal consistent set W such that $\vdash \bigwedge W \to \neg A$, and all of the formulas $t_i{:}(Q_i \vee \neg Q_i)$ belong to W.*

Let now W denote the corresponding set which satisfies the last statement of the lemma. We will construct the model $M = (*, Op, v)$ in the following way. The evaluation $*$ is defined as follows: $S_i^* = 1$ if $S_i \in W$ and $S_i^* = 0$ otherwise.

Recall that for propositional variables $v(S_i) \rightleftharpoons S_i$. Now we are going to define the assignment v for all subformulas and all proof terms in A and then the appropriate operations Op will be written down.

Consider the following order relation on the proof terms which occur in A:

$$t_1 \prec t_2 \text{ iff } t_2{:}D(t_1) \in W \text{ for some formula } D.$$

Recall that W is consistent, so, according to the monotonicity axiom, every chain of the kind $t_1 \succ t_2 \succ \ldots$ is finite.

Let now t be any minimal (with respect to the defined order) proof term such that $v(t)$ is not defined. Then for every formula $t{:}D \in W$ the translation $v(D)$ is already defined. Indeed, either D contains no occurrences of other proof terms (in this case $v(D) = D$), or some other proof terms occur in D, which are less then t with respect to described order, but the translation v is already defined for them, so $v(D)$ is already defined too. Thus, we can define $v(t)$ in the following way:

$$v(t) \rightleftharpoons \{v(D) \mid t{:}D \in W\}. \tag{1}$$

Eventually, the translation v will be defined for all proof terms and all subformulas of A.

Lemma 12 *For every subformula B of the formula A one has*

$$M \models B \textit{ iff } \mathcal{LPM} \vdash \bigwedge W \to B.$$

Proof. Induction on the steps of the described procedure for the definition of the mapping v. On the first step the translation v was defined for the propositional variables. In this case it is sufficient to note that $M \models S_i$ iff $S_i \in W$. According to the properties of W one has $S_i \in W$ iff $\vdash \bigwedge W \to S_i$.

The case when B is a boolean combination is trivial.

Let now B be of kind $t{:}C$. By the induction hypothesis for every formula D_i such that $t{:}D_i \in W$ the translation $v(D_i)$ is already defined and

$$M \models D_i \text{ iff } \mathcal{LPM} \vdash \bigwedge W \to D_i.$$

According to the reflexivity axiom the implication $\bigwedge W \to D_i$ is provable for all such D_i, so all D_i are true in M. Thus, according to (1), the configuration $v(t)$ is correct in M. So,

$$M \models t{:}C \text{ iff } v(C) \in v(t).$$

On the other hand, because of (1) one has $v(C) \in v(t)$ iff $t{:}C \in W$, and, by the properties of W,

$$t{:}C \in W \text{ iff } \mathcal{LPM} \vdash \bigwedge W \to t{:}C.$$

It follows from this lemma that $M \not\models v(A)$.

Let us now complete the construction of the model. The translation v is now defined for all proof terms from A, so, it is defined for all proof variables from A. For all other proof variables we put $v(p)$ to be empty configuration. Now we have to define the operations Op appropriate to the translation v:

(·) $v(t) \cdot v(s) \rightleftharpoons v(t \cdot s)$, if $t \cdot s$ occurs in A,
$\delta_1 \cdot \delta_2 \rightleftharpoons \{\psi \mid \phi \to \psi \in \delta_1 \text{ and } \phi \in \delta_2\}$ for all other pairs of configurations.

(+) $v(t) + v(s) \rightleftharpoons v(t + s)$, if $t + s$ occurs in A,
$\delta_1 + \delta_2 \rightleftharpoons \{\psi \mid \psi \in \delta_1 \text{ or } \phi \in \delta_2\}$ otherwise.

(!) $!(v(t)) \rightleftharpoons v(!t)$, if $!t$ occurs in A,
$!\delta \rightleftharpoons \{[\delta]\psi \mid \psi \in \delta\}$ for all other configurations.

($?_B$) $?_{v(B)}(v(t)) \rightleftharpoons v(?_B t)$, if $?_B t$ occurs in A,
$?_\varphi(\delta) \rightleftharpoons \{\neg[\delta]\varphi\}$ if $\varphi \notin \delta$ and empty configuration otherwise, for all other configurations δ.

Note, that the operations Op defined above are correct in M. Indeed, for those configurations which coincide with images of some proof terms from A it follows from the definition of the set X_A and from the properties of W. For all other configurations it follows from the correctness of the minimal operations.

Thus, the model $M = (*, Op, v)$ is constructed, such that $M \not\models A$.

Corollary 11 *The logic $\mathcal{LPM}$ is decidable.*

Proof. It follows from the given proof that the logic $\mathcal{LPM}$ is complete under the class of finitely defined basic models for which only finite number of propositional variables are true, the operations Op are defined in the minimal way for all arguments except for the finite set, and only finite number of proof variables are interpreted by nonempty configurations.

In fact, the detailed inspection of the proof allows to find the exact upper bounds on the complexity of the countermodel.

4 Arithmetical Completeness

Let $PROOF(x, y)$ denote the following multi-conclusion version of the standard Gödel proof predicate for Peano arithmetic PA:

> "x is a number of a finite set of derivations in PA, and y is a number of a formula, proved by one of this derivations".

It has the following natural properties (in what follows $Th(n)$ stands for the set $\{\varphi \mid PROOF(n, \ulcorner\varphi\urcorner)\}$):

1. For every finite set Ω of theorems of PA there exists a natural number n such that $\Omega = Th(n)$.
2. For any arithmetical formula $\psi(x)$ with one free variable x and any natural numbers m, n the following holds:

$$PA \vdash PROOF(\overline{m}, \ulcorner\psi(\overline{n})\urcorner) \to \overline{m} > \overline{n}.$$

Below, instead of the predicate $PROOF$ we may consider any proof predicate with these two properties.

An arithmetical interpretation f of the language $\mathcal{LPM}$ is organized as follows:

- arithmetical sentences are assigned to all propositional variables;
- computable functions $+', \cdot', !'$ and $?'_\varphi$ on natural numbers are defined in such a way that the corresponding operational properties are satisfied;
- numerals are assigned to all proof variables.

Interpretation f commutes with boolean connectives and operations on proofs and

$$f(?_A(t)) = ?'_{f(A)}(f(t)), \quad f(t{:}A) = PROOF(f(t), \ulcorner f(A)\urcorner).$$

Theorem 2 *Let A be any formula in the language $\mathcal{LPM}$. Then the following holds*

a) if $\mathcal{LPM} \vdash A$ then $PA \vdash f(A)$ for all arithmetical interpretations f;

b) if $\mathcal{LPM} \nvdash A$ then there exists arithmetical interpretation f such that $PA \vdash \neg f(A)$.

Proof. a) Straightforward induction on the length of proof in $\mathcal{LPM}$.

b) Let A be unprovable in $\mathcal{LPM}$. Then there exists a basic model M, such that $M \models \neg A$. It was already mentioned above that we can take M to be finitely defined. First of all we define arithmetical assignment for propositional variables in the following way:

$$f(S_i) \rightleftharpoons \begin{cases} i = i, & \text{if } M \models S_i; \\ i = i + 1, & \text{otherwise}. \end{cases}$$

It is clear that for every pure propositional formula φ one has

$$M \models \varphi \;\Leftrightarrow\; PA \vdash f(\varphi). \tag{2}$$

Firstly, we are going to extend the definition of f to all formulas of the languages L^δ and to keep the condition (2) true.

Let $\delta = \{\varphi_1, \ldots, \varphi_n\}$ be a finite set of L^δ-formulas such that the assignment $f(\varphi_1), \ldots, f(\varphi_n)$ is already defined. If all formulas $f(\varphi_1), \ldots, f(\varphi_n)$ are provable in PA then put $f(\delta)$ to be the least natural number n such that

$$Th(n) = \{f(\varphi_1), \ldots, f(\varphi_n)\}.$$

Otherwise, take $f(\delta)$ to be the least natural number n which is not a Gödel number of any set of proofs, and such that it was not defined to be the f-image for another configuration δ'.

Stipulating that $f([\delta]\varphi) \rightleftharpoons PROOF(\overline{f(\delta)}, \overline{f(\varphi)})$ the interpretation f could be defined for all formulas of the language L^δ. One can easily check that f has the following properties:

1. for every formula φ in the language L^δ the formula $f(\varphi)$ is arithmetical Δ_1-sentence;
2. f is computable;
3. f^{-1} is computable too, namely, for every arithmetical sentence one can effectively either find its pro-image or establish that it is not an image of some L^δ-formula; the same holds for configurations: for every natural number n one can effectively either find the corresponding configuration, or establish that n is not an image of any L^δ-configuration;
4. f is injective, i.e. if $f(\varphi)$ coincides with $f(\psi)$ then φ and ψ coincide too;
5. the condition (2) holds for every formula in the language L^δ;
6. an arbitrary nonempty configuration δ is correct in M if and only if $f(\delta)$ is a Gödel number of some multiple proof in PA.

Now, using the interpretation of the language $\mathcal{LPM}$ into the language L^δ given with the basic model M we can extend definition of f to all formulas of the language $\mathcal{LPM}$.

For proof terms put $f(t) \rightleftharpoons f(v(t))$. The only thing we need to do is an appropriate definition of arithmetical functions $\cdot$, $+$, ! and ?. The properties $2 - -4, 6$ listed above allows us to carry this definition from the model M. We define:

$$\begin{aligned} f(\delta_1) \cdot' f(\delta_2) = f(\delta_3) &\text{ iff } \delta_1 \cdot \delta_2 = \delta_3; \\ f(\delta_1) +' f(\delta_2) = f(\delta_3) &\text{ iff } \delta_1 + \delta_2 = \delta_3; \\ !' f(\delta_1) = f(\delta_2) &\text{ iff } !\delta_1 = \delta_2; \\ ?'_{f(\varphi)} f(\delta_1) = f(\delta_2) &\text{ iff } ?_\varphi \delta_1 = \delta_2. \end{aligned}$$

In case not all of the arguments are images of some configurations define the result in arbitrary computable way.

Since the model M is finitely defined the operations defined in such a way are computable. It is clear that the condition (2) remains true for all formulas of the language $\mathcal{LPM}$. So, $PA \vdash \neg f(A)$. Q.E.D.

5 Protocolling Extensions of Theories

In this section we generalize arithmetical semantics in the following way. Let us fix an arbitrary consistent recursively enumerable theory T. We assume that all tautologies in the language of T are provable in T and the *modus ponens* rule is admissible. Also we assume that T is given by some decidable set of axioms and rules of inference.

Consider now a non-deterministic theorem proving device M_T which works as follows. At each moment the configuration of M_T is characterized by a finite set of formulas in the language of T. The computational process of M_T satisfies the following conditions (below, Ω_i denotes a configuration of M_T at the moment i): M_T starts from the empty set of formulas, so $\Omega_1 = \emptyset$, and there are two ways to obtain Ω_{i+1} from Ω_i:

1. (deduction) $\Omega_{i+1} = \Omega_i \cup \{\varphi\}$, where φ is
   - either a tautology in the language of T,
   - or an axiom of T,
   - or immediate consequence of some formulas from Ω_i by one of the rules of inference;
2. (clearing of memory) $\Omega_{i+1} = \Omega_i \setminus \{\varphi\}$, where $\varphi \in \Omega_i$.

Any finite sequence of configurations $\Omega_1, \ldots, \Omega_n$ which satisfies these conditions will be called *a protocol for* M_T. We say that the given protocol *verifies* formula φ if $\varphi \in \Omega_n$. We say that M_T verifies formula φ if there exists a protocol for M_T which verifies φ.

It is clear that M_T verifies φ if and only if φ is a theorem of T.

Suppose now that M_T is strong enough to analyze its own protocols. Namely, given a list of configurations π and a formula φ it can decide whether π is a protocol for M_T which verifies φ or not (such a procedure exists since the corresponding relation is decidable). Also, it means that the device is able to operate with the set of atomic formulas extended in the following way:

> if π is a list of configurations and φ is a formula then $[\pi]\varphi$ is also a formula denoting the fact that π verifies φ.

Now, the corresponding item in the description of the process of computation looks as follows:

1. (deduction) $\Omega_{i+1} = \Omega_i \cup \{\varphi\}$, where φ is
   - either a tautology in the extended language,
   - or an axiom of T,
   - or immediate consequence of some formulas from Ω_i by one of the rules of inference;
   - or has the form $[\pi]\psi$, where π verifies ψ,
   - or has the form $\neg[\pi]\psi$, where π does not verify ψ.

This advanced version of the device M_T will be denoted by M_T^e.

Such an extension of the proving device described above corresponds to the following extension T^e of the theory T. The language of T^e is an extension of the language of T by the formulas of kind $[\pi]\psi$. The set of axioms is extended by all tautologies in the extended language and formulas of the form $[\pi]\varphi$, where π verifies φ, and $\neg[\pi]\varphi$, where π does not verify φ. It is clear that $T^e \vdash \varphi$ if and only if M_T^e verifies φ.

Now we can define the protocolling semantics for the logic $\mathcal{LPM}$. Let T^e be the protocolling extension of a theory T. An interpretation f of the language of $\mathcal{LPM}$ into the language of T^e is a triple $\langle *, Op, v\rangle$ with the following parameters:

$*$ maps propositional variables to formulas in the language of T^e;
Op is a set of operations $+', \cdot', !'$ and $?'_\varphi$ defined on the set of all protocols of M_T^e which satisfy the corresponding operational axioms of $\mathcal{LPM}$;
v maps every proof variable to a protocol for M_T^e.

Stipulating that f commutes with boolean connectives and operations it could be extended to the set of all formulas and proof terms of the language $\mathcal{LPM}$. In particular,

$$f(?_A(t)) = ?'_{f(A)}(f(t)), \quad f(t{:}A) = [f(t)]f(A).$$

Theorem 3 *Suppose that T is a theory and A is an arbitrary formula in the language of $\mathcal{LPM}$. Then*

a) if $\mathcal{LPM} \vdash A$ then for any interpretation f one has $T^e \vdash f(A)$;
b) if $\mathcal{LPM} \nvdash A$ then there exists an interpretation f such that $T^e \vdash \neg f(A)$.

Sketch of proof. The first proposition of the theorem follows by straightforward checking of all axioms of $\mathcal{LPM}$ to be provable in T^e under any interpretation.

To prove the proposition (b) we use the completeness of $\mathcal{LPM}$ with respect to the basic models. Let M be a basic model such that $M \not\models A$. One can embed M into T^e in the following way. If a propositional variable S_i is true in M then we put $f_M(S_i)$ to be a tautology in the language of T^e, otherwise put $f_M(S_i)$ to be any inconsistent sentence. The only thing one should care about is injectivity of the embedding: different formulas of $\mathcal{LPM}$ should be interpreted by different formulas of T^e. It is clear that for every correct configuration $\delta = \{\varphi_1, \ldots, \varphi_n\}$ the corresponding formulas will be provable in T^e. So, there exists a protocol π_δ for M_T^e which verifies the set of formulas $\{f_M(\varphi_1), \ldots, f_M(\varphi_n)\}$. We put $f_M(\delta) \rightleftharpoons \pi_\delta$. The operations are treated in the obvious way.

Since $M \models \neg A$, one has $T^e \vdash \neg f_M(A)$ for the described interpretation f_M.

6 First Order Case

The study of first order logic of proofs was initiated in [5]. Given an arithmetical theory T and a class $\mathcal{K}$ of proof predicates the logic $\mathcal{QLP}_\mathcal{K}(T)$ is defined as the set of all formulas in the corresponding language described below, which are

provable in T under every arithmetical interpretation based on a proof predicate from $\mathcal{K}$. It was shown there that for different natural classes $\mathcal{K}$ and theories T the corresponding logic $\mathcal{QLP}_{\mathcal{K}}(T)$ is not effectively axiomatizable.

In this section we consider the first order logic $\mathcal{QLP}_{\mathcal{M}}$ and prove arithmetical completeness theorem for it.

6.1 The Main Definition

The language of the first order logic of proofs $\mathcal{QLP}_{\mathcal{M}}$ is the extension of the pure predicate language with infinite set of proof variables $p_1, p_2, \ldots$, proof predicate $t:F$, and symbols of operations $\cdot$, $+$, $!$, $?_A$ (the same as in the propositional case) and infinite set of new unary operational symbols $g_1, g_2, \ldots$, for the operation of generalization over the corresponding individual variables $x_1, x_2, \ldots$.

Formulas and proof terms of the language $\mathcal{QLP}_{\mathcal{M}}$ are constructed as follows:

1. if Q is any n-ary predicate symbol and $y_1, \ldots, y_n$ are individual variables then $Q(y_1, \ldots, y_n)$ is a formula;
2. if A, B are formulas then $\neg A$, $(A \wedge B)$, $(A \vee B)$, $(A \to B)$ are formulas too;
3. if A is a formula, and x_i is an individual variable then $\forall x_i A$ is a formula too, and x_i is excluded from the set of free variables in A;
4. proof variables $p_1, p_2, \ldots$ are proof terms;
5. if s, t are proof terms then $(s \cdot t)$, $(s + t)$, $!t$, $g_i(t)$ are proof terms too;
6. if t is a proof term and A is a formula then $?_A(t)$ is a proof term;
7. if A is a formula, t is a proof term then $(t : A)$ is a (quasiatomic) formula with no free variables, i.e. the proof operator bounds all free occurrences of individual variables in A.

Definition 3 *The logic $\mathcal{QLP}_{\mathcal{M}}$ is defined by the following axioms (over the first order calculus in the extended language).*

Operational axioms:

1. $s:(A \to B) \to (t:A \to (s \cdot t):B)$ *(composition),*
2. $(t:A) \vee (s:A) \to (t + s):A$ *(non-deterministic choice),*
3. $t:A \to !t:(t:A)$ *(verification),*
4. $\neg t:A \to (?_A t):(\neg t:A)$ *(negative introspection),*
5. $t:A \to g_i(t):\forall x_i A$ *(generalization),*

General axioms:

6. $t:A \to A$ *(reflexivity),*
7. $\neg(t_1:A_1(t_2) \wedge t_2:A_2(t_3) \wedge \ldots \wedge t_n:A_n(t_1))$ *(monotonicity).*

The rules of inference: $\dfrac{A, A \to B}{B}$ *(modus ponens),* $\quad \dfrac{A}{\forall x_i A}$ *(generalization).*

6.2 Arithmetical Semantics

Let T be any consistent extension of PA with decidable set of axioms. We suppose that Gödel numbering of arithmetical language is fixed, so for every such a theory we may consider multi-conclusion version of the standard proof predicate:

"x is a number of a finite set of derivations in T, and y is a number of a formula, proved by one of these derivations".

The corresponding arithmetical formula is denoted by $PROOF_T(x, y)$.

An arithmetical interpretation f_T of the language $\mathcal{QLP}_{\mathcal{M}}$ is organized in the following way:

- every atomic formula is interpreted by an arithmetical formula with the same set of free variables;
- f_T commutes with the Boolean connectives, quantifiers and substitution of variables, i.e. if $f_T(Q(x)) = \varphi(x)$ then $f_T(Q(y)) = \varphi(y)$;
- every proof variable is interpreted by a natural number;
- totally computable functions $\cdot', +', !', ?'_\varphi$ and g'_i for all i and φ are defined, which satisfy the operational axioms 1–5 of the logic $\mathcal{QLP}_{\mathcal{M}}$, i.e.
  - if $PROOF_T(n, \lceil \varphi \to \psi \rceil)$ and $PROOF_T(m, \lceil \varphi \rceil)$ then $PROOF_T(n \cdot' m, \lceil \psi \rceil)$;
  - if not $PROOF_T(n, \lceil \varphi \rceil)$ then $PROOF_T(?'_\varphi(n), \lceil \neg PROOF_T(n, \lceil \varphi \rceil) \rceil)$;
  - if $PROOF_T(n, \lceil \varphi \rceil)$ then $PROOF_T(g'_i(n), \lceil \forall x_i \varphi \rceil)$ etc.

  for any natural numbers i, m, n and arithmetical formulas φ and ψ ;
- using these functions the interpretation of proof terms is computed;
- $f_T(t{:}A) \rightleftharpoons PROOF_T(\overline{f_T(t)}, \overline{\lceil f_T(A) \rceil})$.

6.3 Arithmetical Completeness

Theorem 4 *Let A be any formula in the language $\mathcal{QLP}_{\mathcal{M}}$. Then*

a) if $\mathcal{QLP}_{\mathcal{M}} \vdash A$ then for every theory T and interpretation f_T one has $T \vdash f_T(A)$;

b) if $\mathcal{QLP}_{\mathcal{M}} \nvdash A$ then there exists an extension T of PA by finite set of true sentences and interpretation f_T such that $T \nvdash f_T(A)$.

Sketch of proof. a) Straightforward induction on the proof in $\mathcal{QLP}_{\mathcal{M}}$.

b) Scheme of the proof is the following. Firstly, we build finite set X_A of quasiatomic formulas which are adequate to A. Using this set we turn from the fact that $\mathcal{QLP}_{\mathcal{M}} \nvdash A$ to the fact of nonprovability of the implication $K_A^* \to A^*$ in the first order calculus for the conjunction K_A and translation $*$ described below. According to the arithmetical formalization of the Gödel completeness theorem there exists arithmetical interpretation h of the pure predicate language such that $h(K_A^* \to A^*)$ is false in the standard model $\mathbf{N}$ of arithmetic. Then, by combining the translation $*$ and interpretation h we construct the desired theory T and interpretation f_T.

Let $Tm(A)$ denote the set of all proof terms which occur in A. The set X_A is defined as the minimal set of quasiatomic formulas such that

1. all quasiatomic subformulas of A belong to X_A;
2. if $(s \cdot t) \in Tm(A)$ and $s{:}(B \to C) \in X_A$, $t{:}C \in X_A$ then $s \cdot t{:}C \in X_A$;
3. if $(s + t) \in Tm(A)$ and either $s{:}B \in X_A$, or $t{:}B \in X_A$ then $s + t{:}B \in X_A$;
4. if $(!t) \in Tm(A)$ and $t{:}B \in X_A$ then $!t{:}t{:}B \in X_A$;

5. if $g_i(t) \in Tm(A)$ and $t{:}B \in X_A$ then $g_i(t){:}(\forall x_i B) \in X_A$;
6. if $?_B(t) \in Tm(A)$ then $?_B(t){:}(\neg t{:}B) \in X_A$.

One can show by induction of the length of a proof term t that for every $t \in Tm(A)$ the set $\{A \mid t{:}A \in X_A\}$ is finite. So, X_A is finite too.

Let K_A denote the conjunction of all proper axioms of $\mathcal{QLP}_{\mathcal{M}}$ which use only formulas from X_A. For axiom 6 we take its universal closure.

Consider now a translation $*$ of the language $\mathcal{QLP}_{\mathcal{M}}$ into the pure predicate language which substitutes corresponding fresh propositional variables $S_{t:F}$ for all the occurrences of quasiatomic formula $t{:}F$.

It leaves unchanged all pure predicate formulas, and every quasiatomic formula $t : F$ is interpreted by some fresh propositional variable, say $S_{t:F}$. We stipulate that $*$ commutes with Boolean connectives, quantifiers and operation of renaming of individual variables.

One can easily verify that implication $K_A^* \to A^*$ is not provable in the first order calculus, since $\mathcal{QLP}_{\mathcal{M}} \nvdash A$. So, according to the arithmetical formalization of the Gödel completeness theorem for the first order calculus (see [8]), there exists an arithmetical interpretation h of the pure predicate language, such that the formula $h(K_A^* \to A^*)$ is false in $\mathbf{N}$, i.e. $\mathbf{N} \models h^*(K_A) \wedge \neg h^*(A)$.

The composition of $*$ and h (denoted by h^*) is arranged very similar to the desirable interpretation f_T. Indeed, it assigns to every atomic predicate formula some arithmetical formula with the same set of free variables, it commutes with boolean connectives, quantifiers and operation of renaming variables, and $\mathbf{N} \models \neg h^*(A)$. Unfortunately, h^* does not correlate with provability, but it can be fixed.

Without loss of generality one may assume that for every propositional variable S either $h(S) = \top$ or $h(S) = \bot$ for $\top \rightleftharpoons (0 = 0)$, $\bot \rightleftharpoons (0 = 1)$.

Recall that for every quasiatomic formula $t : F$ from X_A the implication $t{:}F \to F$ is included into K_A. So, if $h^*(t{:}F) = \top$ then $\mathbf{N} \models h^*(F)$.

Now we can define the desired extension T of PA:

$$T = PA + \{h^*(F) \mid t{:}F \in X_A \text{ and } h^*(t{:}F) = \top\}.$$

It follows from the observation above that all additional axioms are true in $\mathbf{N}$.

We define $f_T(Q) \rightleftharpoons h^*(Q)$ for all pure predicate formulas. For all proof variables not occurring in A put $f_T(p) \rightleftharpoons 0$. Now we are going to define interpretation f_T for all proof terms from $Tm(A)$, and then the corresponding computable functions for the operations on proofs will be provided.

Consider a partial order on $Tm(A)$ which is the transitive closure of the following relation: $t_1 \prec t_2$ iff for some formula $B(t_1)$ one has $t_2{:}B(t_1) \in X_A$ and $h^*(t_2{:}B(t_1)) = \top$. This order is well founded, since the monotonicity axiom for all such formulas is included in K_A.

Consider a term t, which is minimal with respect to this order relation. The set $\Omega(t) \rightleftharpoons \{F \mid t : F \in X_A \text{ and } h^*(t : F) = \top\}$ contains only pure predicate formulas, therefore the interpretation f_T is already defined for them. On the other hand, according to the definition of T the results of translation of all these formulas are axioms of T. So, there are infinitely many proofs of the set $f_T(\Omega(t))$.

We define $f_T(t)$ to be Gödel number of one of them. The only thing one should care about is the injectivity of f_T on the set $Tm(A)$. It will be needed for the appropriate definition of operations.

Thus, f_T is defined for all minimal proof terms. One can easily see that for every formula F in the language $\mathcal{QLP}_{\mathcal{M}}$ if $f_T(F)$ is already defined then the following condition holds:

$$PA \vdash f_T(F) \leftrightarrow h^*(F) \tag{3}$$

Let now $t \in Tm(A)$ be a minimal proof term such that $f_T(t)$ is not defined. Then, for every formula $F \in \Omega(t)$ the interpretation $f_T(F)$ is already defined. On the other hand, according to property (3) all formulas from $f_T(\Omega(t))$ are equivalent in PA to some axioms of T. So, they are provable in T, and we can define $f_T(t)$ as Gödel number of a proof of this set of formulas. It is clear, that for the extended version of f_T the condition (3) remains true. In a finite number of steps the definition of f_T can be extended to all proof terms from $Tm(A)$. Then, following this definition one easily can define the appropriate computable functions for operations of proofs (see the corresponding part in the proofs of theorems 1 and 2).

So, according to the condition (3) one has $\mathbf{N} \models \neg f_T(A)$, since $\mathbf{N} \models \neg h^*(A)$. Hence, $T \nvdash f_T(A)$.

Acknowledgments

I am very grateful to my scientific advisor and coordinator professor Sergei Nikolaevich Artemov for his permanent care and encouragement. Also, I am very thankful to Vladimir Nikolaevich Krupski and Tanya Yavorskaya (Sidon) for helpful discussion of the subject.

References

1. S. Artemov, *Logic of Proofs.* Annals of Pure and Applied Logic, **67** (1994), pp. 29–59.
2. S. Artemov, *Explicit provability: the intended semantics for intuitionistic and modal logic.* Techn. Rep. No 98–10. Mathematical Science Institute, Cornell University, 1998. Available at http://www.math.cornell.edu/~ artemov/publ.html
3. S. Artemov, *On explicit reflection in theorem proving and formal verification.* In Springer Lecture Notes in Artificial Intelligence, **1632** (1999), pp. 267–281.
4. S. Artemov, E. Kazakov and D. Shapiro, *Logic of knowledge with justifications.* Techn. Rep. No 99–12. Mathematical Science Institute, Cornell University, 1999.
5. S. Artemov and T. Sidon-Yavorskaya, *On the first order logic of proofs.* Techn. Rep. No 99–11. Mathematical Science Institute, Cornell University, 1999.
6. K. Gödel, *Eine Interpretation des intuitionistischen Aussagenkalkuls,* Ergebnisse Math. Colloq., Bd. 4 (1933), S. 39–40.
7. K. Gödel, *Vortrag bei Zilsel (1938),* in S. Feferman, ed., Kurt Gödel Collected Works. Volume III, Oxford University Press, 1995
8. D. Hilbert, P. Bernays, Grundlagen der Mathematik, II, Springer–Verlag, 1968.

9. A. Kolmogoroff, *Zur Deutung der intuitionistischen Logik,* Math. Ztschr., Bd. 35 (1932), S. 58–65.
10. A. Mkrtychev, *Models for the Logic of Proofs.* In Springer Lecture Notes in Computer Science, **1234** (1997), pp. 266–275.

Author Index